Introduction

Sudakshina Dutta

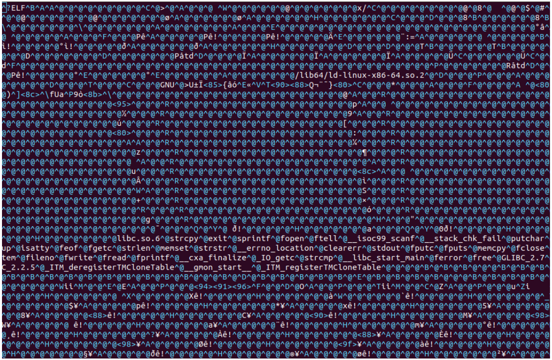
IIT Goa

18^{th} January, 2022

A code segment

int *func* (int n_1, int n_2)
{
    int i;
    for($i = 1$; $i \leq n_1$ && $i \leq n_2$; $++i$)
    {
        if($n_1 \% i == 0$ && $n_2 \% i == 0$)
            $result = i$;
    }
    return $result$;
}

Can you tell what is this ?



- All of us have written computer programs in high-level languages e.g., C, C++, Java, etc.
- Although we write programs in high-level languages, computer executes the programs written in machine language
- Programming in machine language requires memorization of the binary codes — difficult for program-writers
- Hence, the requirement of **Compilers**

Introduction

- A Compiler is a program that can read a program in one language (source) and translate it into an equivalent program in another language (target)

- We use compilers for generating machine language program from the input high-level language program

Is translation easy ?

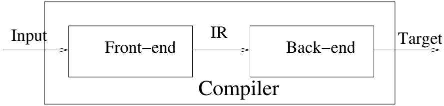
Consider this

- I am standing here “to get” the bus
- I am trying “to get” the gist of the document
- I am trying “to get” some food

The compiler must preserve the meaning of the program being compiled.

Translation

- A compiler must both understand the source program that it takes as input and map its functionality to the target machine



- The front-end focuses on understanding the source-language program
- An *intermediate representation* is generated
- The back-end focuses on mapping programs to the target machine

How to understand the input ?

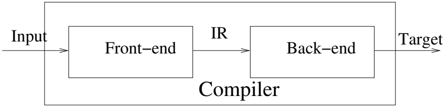
- To understand the input program, syntax has to be checked and IR has to be generated
  - The syntax of the source language is defined by some finite set of rules, called a grammar
- The compiler must compare the program's structure against a definition for the language
  - For example,in English, many sentences have the form
    Sentence $\rightarrow$ *Subject* verb *Object* endmark
    Any sentence e.g., "Compilers are softwares." is an example

How to understand the input ?

- Similarly, in programming language,
  - $S \rightarrow V = E$
  - $E \rightarrow E + F$
  - $E \rightarrow F$
  - $F \rightarrow F * T$
  - $F \rightarrow T$
  - $T \rightarrow int$
  - $V \rightarrow int$
- Apply for any expression "a = b + c*d"
- The expression "b + c*d" is generated as follows. Derivation should start from E
  - $E \rightarrow E + F$
  - $E \rightarrow E + F * T$
  - $E \rightarrow F + F * T$
  - $E \rightarrow T + T * T$
  - $E \rightarrow int + int * int$
- The derivation shows "b + c*d" can be generated from E

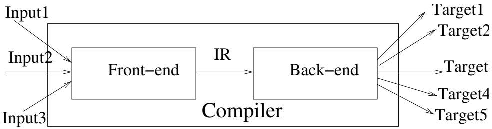
Translation

- After checking the syntax, the front-end must encode its knowledge of the source program in some structure for later use by the back-end
- The structure is called intermediate representation (IR)
- The front-end must ensure that the source program is well-formed, and it must map that code into their IR



Advantage

- We may have many input languages and many machines where the target programs may run
- If there is no IR, then we need 15 compilers
- If IR is present, then we need to only construct 3 frond-ends and 5 back-ends



Conversion into IR

- Compilers use a variety of different kinds of IR
  - An example - three address code
- IR representation of "a = b + c*d ;"
  - $t_0 \leftarrow c * d$
  - $t_1 \leftarrow b + t_0$
  - $a \leftarrow t_1$

Conversion to machine code

We need to convert the following to machine code

- $t_0 \leftarrow c * d$
- $t_1 \leftarrow b + t_0$
- $a \leftarrow t_1$

Converted form

- MOV c, $R0$
- MUL d, $R0$
- MOV $R0$, $R1$
- MOV b, $R0$
- ADD $R1$, $R0$
- MOV $R0$, a

Why study compilers ?

- It helps us learn to develop theory of programming languages
- It makes practical use of many algorithms
  - greedy algorithms (register allocation), heuristic search techniques(list scheduling), graph algorithms (dead-code elimination), dynamic programming (instruction selection), finite automata and push-down automata(scanning and parsing), and fixed-point algorithms (data-flow analysis).
- It is generally a software consisting of thousands of lines of code organized into multiple sub-modules
  - The design and implementation of a compiler is a substantial exercise in software engineering
- The knowledge of front-end is very useful

Optimization

Consider the following expression $a \leftarrow b + c * d + c * d$

- $t_0 \leftarrow c * d$
- $t_1 \leftarrow c * d$
- $t_2 \leftarrow b + t_0$
- $t_3 \leftarrow t_2 + t_1$
- $a \leftarrow t_3$

Is there any scope to improve the code ?

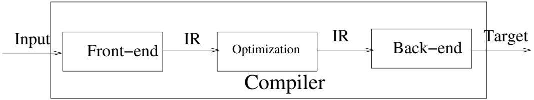
Optimization

Consider the following expression $a \leftarrow b + c * d + c * d$

- $t_0 \leftarrow c * d$
- $t_2 \leftarrow b + t_0$
- $t_3 \leftarrow t_2 + t_0$
- $a \leftarrow t_3$

This optimization is called "common sub-expression elimination" optimization

Optimization



Textbooks

- **Compilers (Second Edition) Principles, Techniques, Tools** by by Aho, Lam, Sethi, Ullman
- **Engineering a Compiler (Second Edition)** by Keith D. Cooper and Linda Torczon

Examinations

- Class test (10)
- Mid-sem (30)
- Class test (10)
- End-sem (40)
- Attendance and class participation (10)

Google classroom code : vatui46