Module 02

Das

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

Summary

Module 02: CS-1319-1: Programming Language Design and Implementation (PLDI)

Overview: Phases of a Compiler

Partha Pratim Das

Department of Computer Science
Ashoka University

ppd@ashoka.edu.in, partha.das@ashoka.edu.in, 9830030880

September 04 & 05, 2023

Objectives & Outline
Phases of a Compiler
C Compilation
Front-end
Lexical Analysis
Syntax Analysis
Semantic Analysis
Intermediate Code Generator
Code Optimization
Back-end
Code Optimization
Target Code Generation
Sample Translation
Summary

Module Objectives

- Understand the phases of a compiler

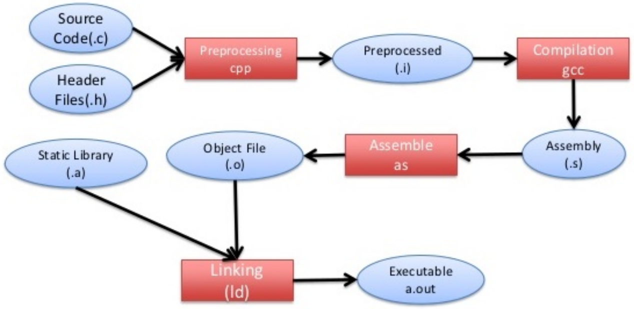
Module Outline

1. Objectives & Outline
2. Phases of a Compiler
   - Overview of Compilation Process
   - Compiler Front-end
     - Lexical Analysis
     - Syntax Analysis
     - Semantic Analysis
     - Intermediate Code Generator
     - Code Optimization
   - Compiler Back-end
     - Code Optimization
     - Target Code Generation
3. Sample Translation
4. Summary

Objectives & Outline
Phases of a Compiler
C Compilation
Front-end
Lexical Analysis
Syntax Analysis
Semantic Analysis
Intermediate Code Generator
Code Optimization
Back-end
Code Optimization
Target Code Generation
Sample Translation
Summary

Build Pipeline: Compiling a C Program

- C Pre-Processor (CPP)
- C Compiler
- Assembler
- Linker



Compilation Flow Diagrams for gcc

Source: http://www.slideshare.net/Bletchley131/compilation-and-execution(slide#2) Broken 12-Sep-2022

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

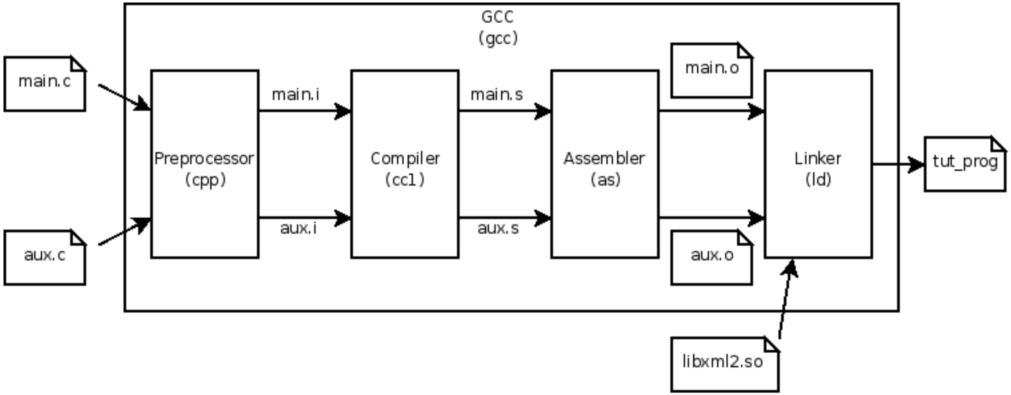
Code Optimization

Target Code Generation

Sample Translation

Summary

Build Pipeline: Compiling a C Program



Source: GNU Compiler Collection, Wikiwand Accessed 12-Sep-2022

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

Summary

Build Pipeline: Compiling a C Program

- The **C preprocessor (CPP)** has the ability for the inclusion of header files, macro expansions, conditional compilation, and line control. It works on `.c`, `.cpp`, and `.h` files and produces `.i` files
- The **Compiler** translates the pre-processed C/C++ code into assembly language, which is a machine level code in text that contains instructions that manipulate the memory and processor directly. It works on `.i` files and produces `.s` files
- The **Assembler** translates the assembly program to binary machine language or object code. It works on `.s` files and produces `.o` files
- The **Linker** links our program with the pre-compiled libraries for using their functions and generates the executable binary. It works on `.o` (static library), `.so` (shared library or dynamically linked library), and `.a` (library archive) files and produces `a.out` file

File extensions mentioned here are for GCC running on Linux. These may vary on other OSs and for other compilers. Check the respective documentation for details. The build pipeline, however, would be the same.

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

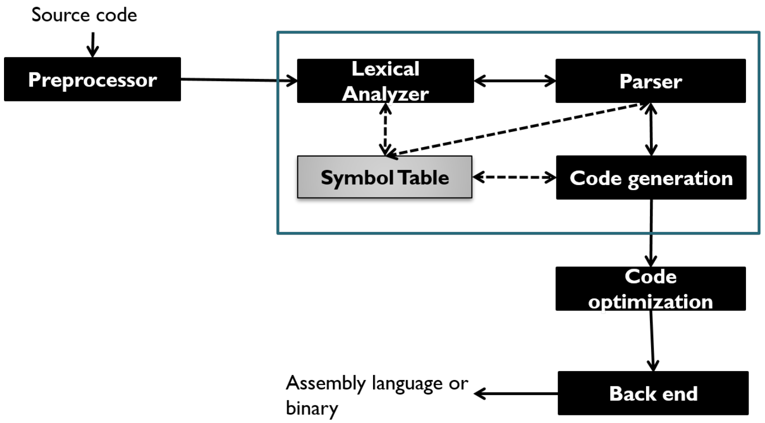
Summary

Common File Extensions

File Type	Linux	Windows
C Source File	.c	.c
C++ Source File	.cc, .cpp	.cpp, .cxx
C/C++ Header File	.h	.h
Pre-processor Output File	.i	.i
Assembly File	.s	.asm
Object File	.o	.obj
Static Library File	.a	.lib
Dynamic Library File	.so	.dll
Executable / Binary File	-	.exe

Objectives & Outline
Phases of a Compiler
C Compilation
Front-end
Lexical Analysis
Syntax Analysis
Semantic Analysis
Intermediate Code Generator
Code Optimization
Back-end
Code Optimization
Target Code Generation
Sample Translation
Summary

Build Pipeline: Compiling a C Program



Four Pass Compiler

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

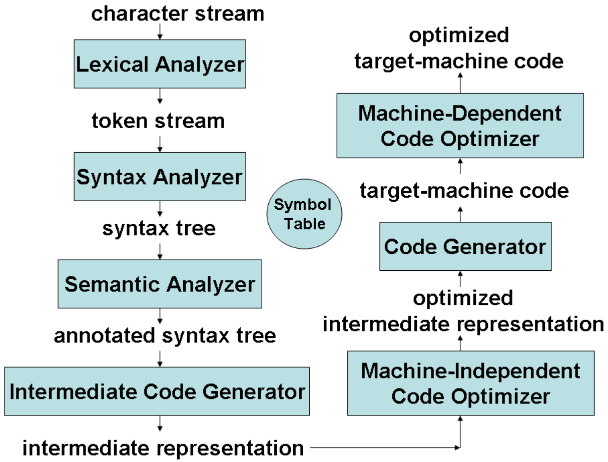
Code Optimization

Target Code Generation

Sample Translation

Summary

Build Phases


character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
annotated syntax tree
Intermediate Code Generator
intermediate representation
Symbol Table
Machine-Independent Code Optimizer
optimized intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
optimized target-machine code

Source: Y N Srikant (NPTEL)

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

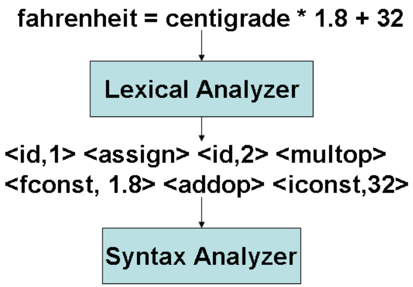
Code Optimization

Target Code Generation

Sample Translation

Summary

Lexical Analysis Phase


fahrenheit = centigrade * 1.8 + 32
Lexical Analyzer
<id,1> <assign> <id,2> <multop>
<fconst, 1.8> <addop> <iconst,32>
Syntax Analyzer

$$\begin{aligned} \textit{fahrenheit} &= \textit{centigrade} * 1.8 + 32 \\ \textit{totalAmount} &= \textit{principalAmount} * 10 + \textit{principalAmount} \\ \textit{finalVelocity} &= \textit{acceleration} * \textit{time} + \textit{initialVelocity} \end{aligned}$$

Source: Y N Srikant (NPTEL)

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

Summary

Lexical Analysis Phase

$$
\begin{aligned}
f &= c * 1.8 + 32 \\
b &= a * 10 + a \\
v &= a * t + u
\end{aligned}
$$

$$
\begin{aligned}
id &= id * num + num \\
id &= id * num + id \\
id &= id * id + id
\end{aligned}
$$

$$
\begin{aligned}
E &= E * E + E \\
(E &= ((E * E) + E))
\end{aligned}
$$

Module 02

Das

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

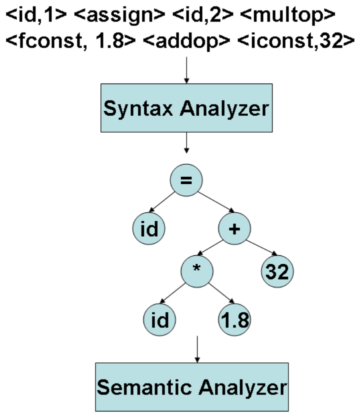
Code Optimization

Target Code Generation

Sample Translation

Summary

Syntax Analysis Phase


<id,1> <assign> <id,2> <multop>
<fconst, 1.8> <addop> <iconst,32>
Syntax Analyzer
=
id
+
*
32
id
1.8
Semantic Analyzer

Source: Y N Srikant (NPTEL)

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

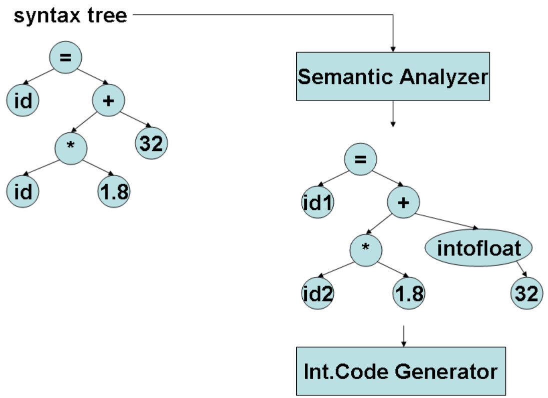
Code Optimization

Target Code Generation

Sample Translation

Summary

Semantic Analysis Phase

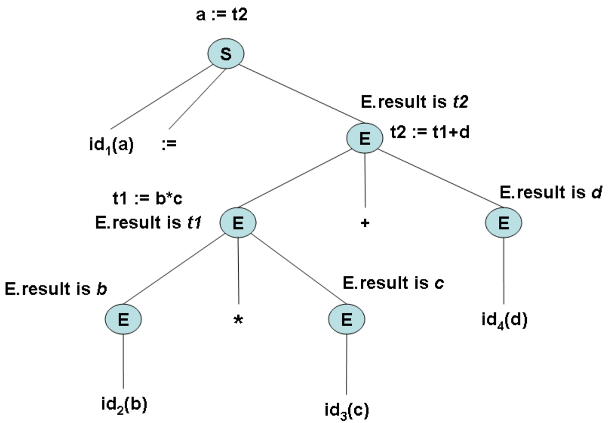


Source: Y N Srikant (NPTEL)

Objectives & Outline
Phases of a Compiler
C Compilation
Front-end
Lexical Analysis
Syntax Analysis
Semantic Analysis
Intermediate Code Generator
Code Optimization
Back-end
Code Optimization
Target Code Generation
Sample Translation
Summary

Intermediate Code Generator: Grammar

$$
\begin{array}{lcl}
G & = & < T, N, S, P > \\
T & = & Set\ of\ terminals \\
 & = & \{id, :=, +, *\} \\
N & = & Set\ of\ non-terminals \\
 & = & \{S, E\} \\
S & = & Start\ Symbol \\
P & = & Set\ of\ productions \\
 & = & \{ \\
S & \rightarrow & id := E \\
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & id \\
 & & \}
\end{array}
$$



Source: Y N Srikant (NPTEL)

Module 02

Das

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

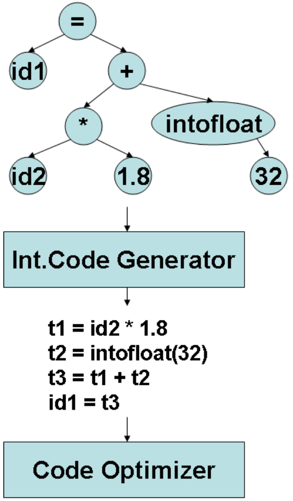
Target Code Generation

Sample Translation

Summary

Intermediate Code Generator: Expression Quad

- Every node denotes a computation
- Computed Expressions are stored in temporary
- Every computation involves 3 addresses max
- Called 3-Address Code (TAC)
- Each TAC is represented as a quad with the operator



```
t1 = id2 * 1.8
t2 = intofloat(32)
t3 = t1 + t2
id1 = t3
```

Source: Y N Srikant (NPTEL)

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

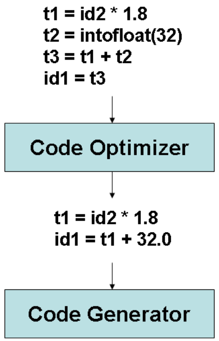
Code Optimization

Target Code Generation

Sample Translation

Summary

Code Optimization



Source: Y N Srikant (NPTEL)

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

Summary

Code Generation and Optimization: Practice Example

- A+B*C+D

- t0=A
- t1=B
- t2=C
- t3=t1*t2
- t4=t0+t3
- t5=D
- t6=t4+t5

- t0=A
- t1=B
- t2=C
- t1=t1*t2
- t0=t0+t1
- t1=D
- t0=t0+t1

- t0=A
- t1=B
- t1=t1*C
- t1=t0+t1
- t1=t1+D

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

Summary

Target Code Generation

- Data Flow and Control Flow Analysis
- Registration Allocation and Assignment
- Code Generation

Module 02

Das

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

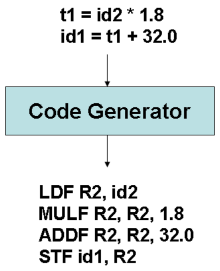
Code Optimization

Target Code Generation

Sample Translation

Summary

Target Code Generation



Source: Y N Srikant (NPTEL)

Module 02

Das

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

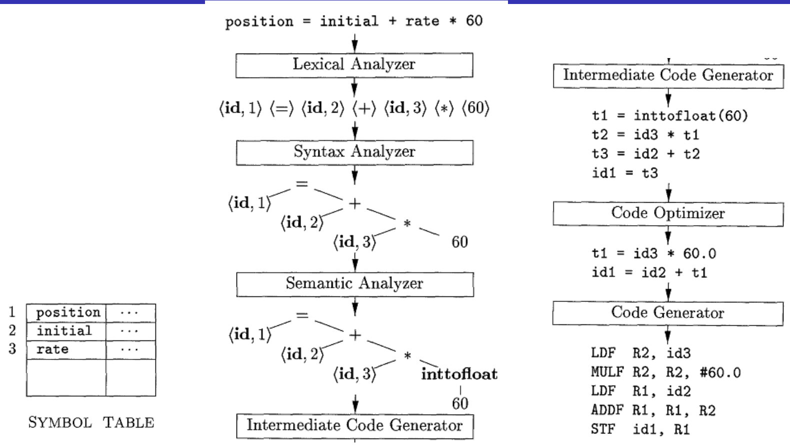
Code Optimization

Target Code Generation

Sample Translation

Summary

Sample pass through Phases



```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

```
t1 = id3 * 60.0
id1 = id2 + t1
```

```
LDF  R2, id3
MULF R2, R2, #60.0
LDF  R1, id2
ADDF R1, R1, R2
STF  id1, R1
```

Source: Dragon Book

Figure: Translation of an assignment statement

Objectives & Outline
Phases of a Compiler
C Compilation
Front-end
Lexical Analysis
Syntax Analysis
Semantic Analysis
Intermediate Code Generator
Code Optimization
Back-end
Code Optimization
Target Code Generation
Sample Translation
Summary

Sample Translation

```
{
    int i; int j;
    float a[100]; float v; float x;

    while (true) {
        do i=i+1; while(a[i]<v);
        do j=j-1; while(a[j]>v);
        if (i>=j) break;
        x=a[i]; a[i]=a[j]; a[j]=x;
    }
}
```

```
01: i = i + 1
02: t1 = a [ i ]
03: if t1 < v  goto 01
04: j = j - 1
05: t2 = a [ j ]
06: if t2 > v  goto 04
07: ifFalse i >= j goto 09
08: goto 14
09: x = a [ i ]
10: t3 = a [ j ]
11: a [ i ] = t3
12: a [ j ] = x
13: goto 01
14: .
```

Module 02

Das

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

Summary

A Typical Compiler Techniques

Promote high level languages by minimizing the execution overhead

Support HPC systems

Compiler

Support several source languages

Potential to translate correctly infinite set of programs written in the source language.

Support several target machines

Collection of compilers

Software engineering techniques

Generate optimal target code from source program ??

Module 02

Das

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

Summary

Languages by Translation Types

Language	**Compilation**	**Typing**	**Framework**
C	Static	Weak[1], Static	No
C++	Static	Strong[2], Static[3]	No[4]
Java	Static	Strong, Static[5]	Yes[6]
Python	Dynamic[7]	Strong, Dynamic	Yes[8]

[1] For example, `void*` breaking typing

[2] If typical C features are not used

[3] Dynamic w/ Polymorphism

[4] RTTI for `dynamic_cast`

[5] Dynamic w/ Polymorphism

[6] Java Virtual Machine – JVM

[7] Interpreter

[8] Python Virtual Machine – PVM

Objectives & Outline

Phases of a Compiler

C Compilation

Front-end

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Generator

Code Optimization

Back-end

Code Optimization

Target Code Generation

Sample Translation

Summary

Module Summary

- Outline of C Compilation Process
- Brief discussion on Phases of a Compiler to understand
  - Front-end flow: Language to TAC
  - Back-end flow: TAC to Machine
- Outline of languages with translation types