Introduction To Compiler

Course Code: CSC 3220 Course Title: Compiler Design

Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	**3**	**Week No:**	**3**	**Semester:**	**Summer**
Lecturer:	*Md Masum Billah; billah.masumcu@aiub.edu* *Masum-billah.net*				

Lecture Outline

1. Phases of a Compiler
2. Practice on Different Input Expressions
3. Linker and Loader
4. Front end and Back end of a compiler
5. Symbol Table Management
6. Error Handler

Objectives and Outcomes

Objectives:

- Understand the Structure of a compiler
- Understand the tools involved(Scanner generator, Parser generator, etc)

Outcome:

- Students will be able to represent the simulation of all phases of a compiler for inputs.

The Phases of a Compiler

Intermediate Code generator: After syntax and semantic analysis of the source program, many compilers generate an explicit low-level or machine-like intermediate representation, which we can think of as a program for an abstract machine. This intermediate representation should have two important properties:

- Easy to Produce and
- Easy to translate into target program

The intermediate representation can have a variety of forms. In this course we consider an intermediate form called " **three address code**".

The Phases of a Compiler

We need to follow some steps to generate three address code.

- Each three address instruction has at most one operator on the right side.
- The compiler must generate a temporary name to hold the value computed by each instruction.
- Some three address instructions have fewer than three operands.

So the output of the intermediate code generator will be

```
temp1 := inttofloat(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

The Phases of a Compiler

Code Optimizer: The machine-independent code-optimization phase attempts to improve the intermediate code so that better target code will result.

- Find More Efficient Ways to Execute Code
- Replace Code With More Optimal Statements
- Significantly improve the running time of the target program

So this phase optimized the code and produced the output as follows

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

The Phases of a Compiler

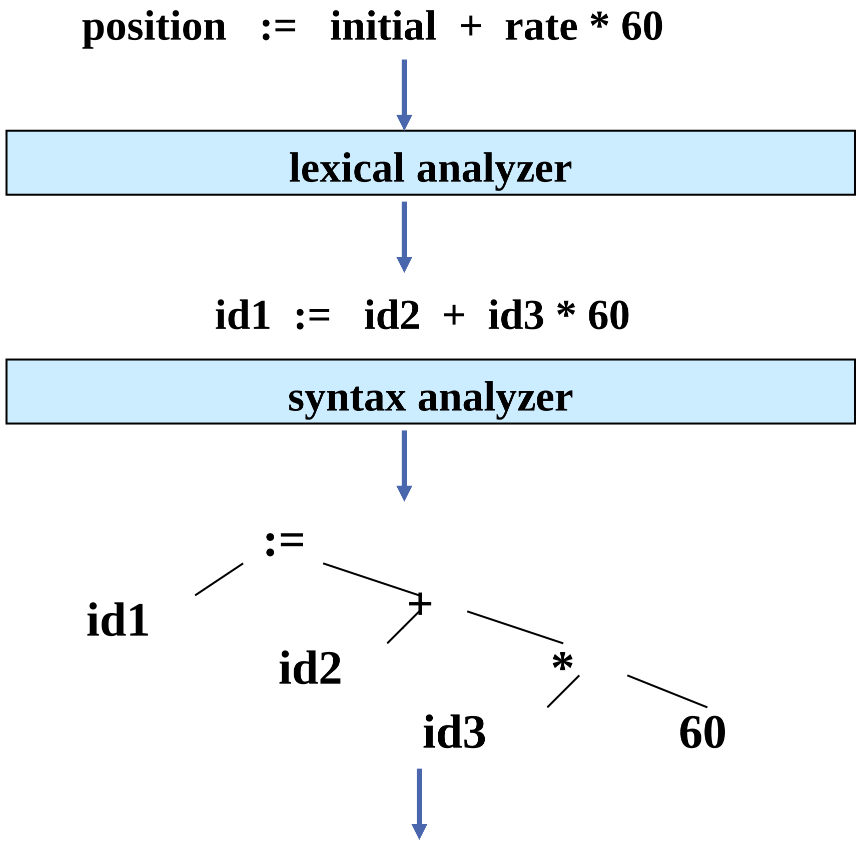
Code Generator: The final phase of the compiler is to generate code for a specific machine. In this phase we consider:

- memory management
- register assignment

The output from this phase is usually assembly language or relocatable machine code.

```
MOVF R2,id3
MULF R2,#60.0
MOVF R1,id2
ADDF R1, R2
MOVF id1,R1
```

Reviewing the Entire Process



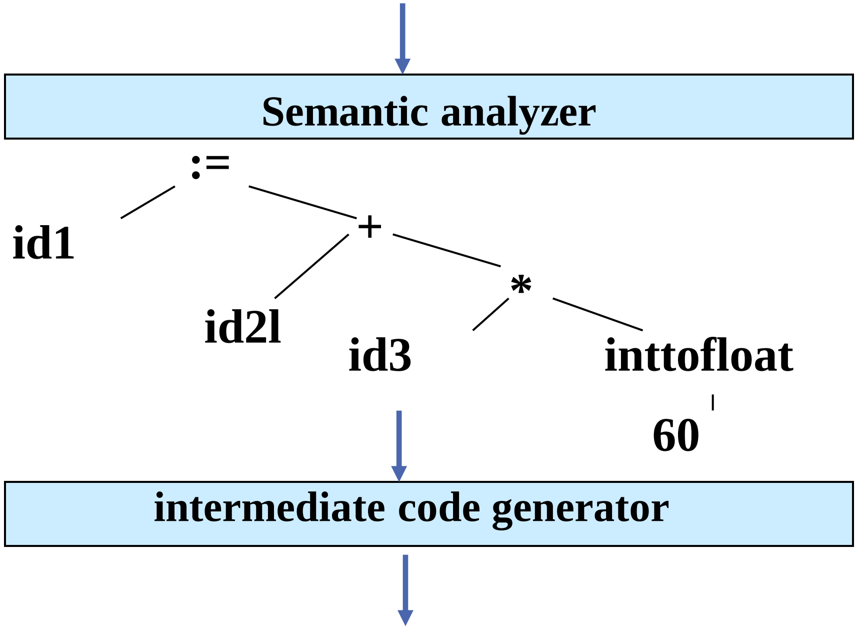
position := initial + rate * 60

lexical analyzer

id1 := id2 + id3 * 60

syntax analyzer

Reviewing the Entire Process



Semantic analyzer

:=

id1

+

id2l

*

id3

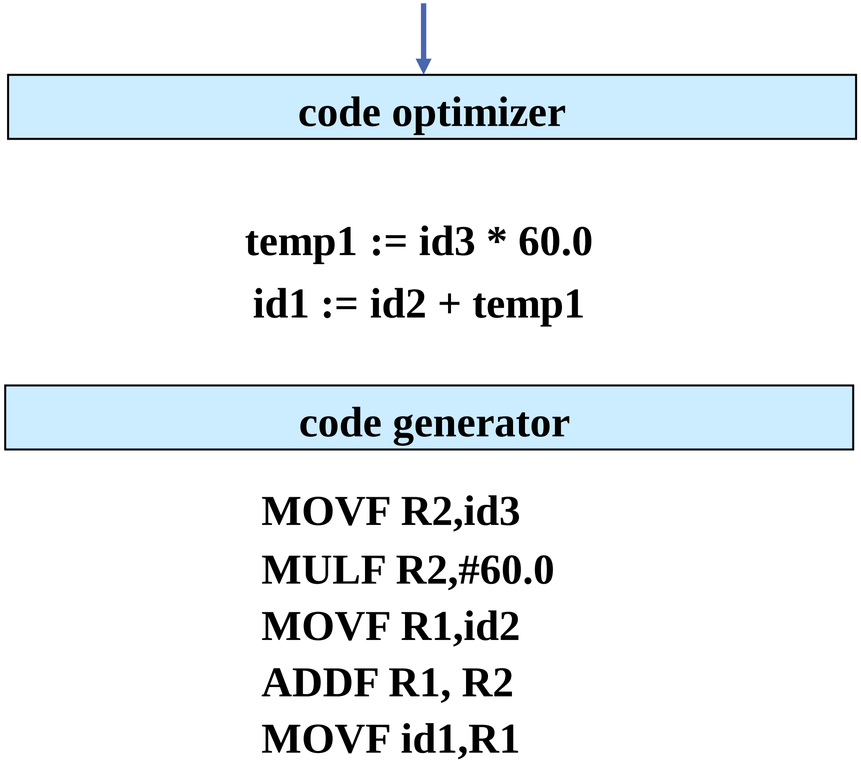
inttofloat

60

intermediate code generator

```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

Reviewing the Entire Process



code optimizer

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

code generator

```
MOVF R2,id3
MULF R2,#60.0
MOVF R1,id2
ADDF R1, R2
MOVF id1,R1
```

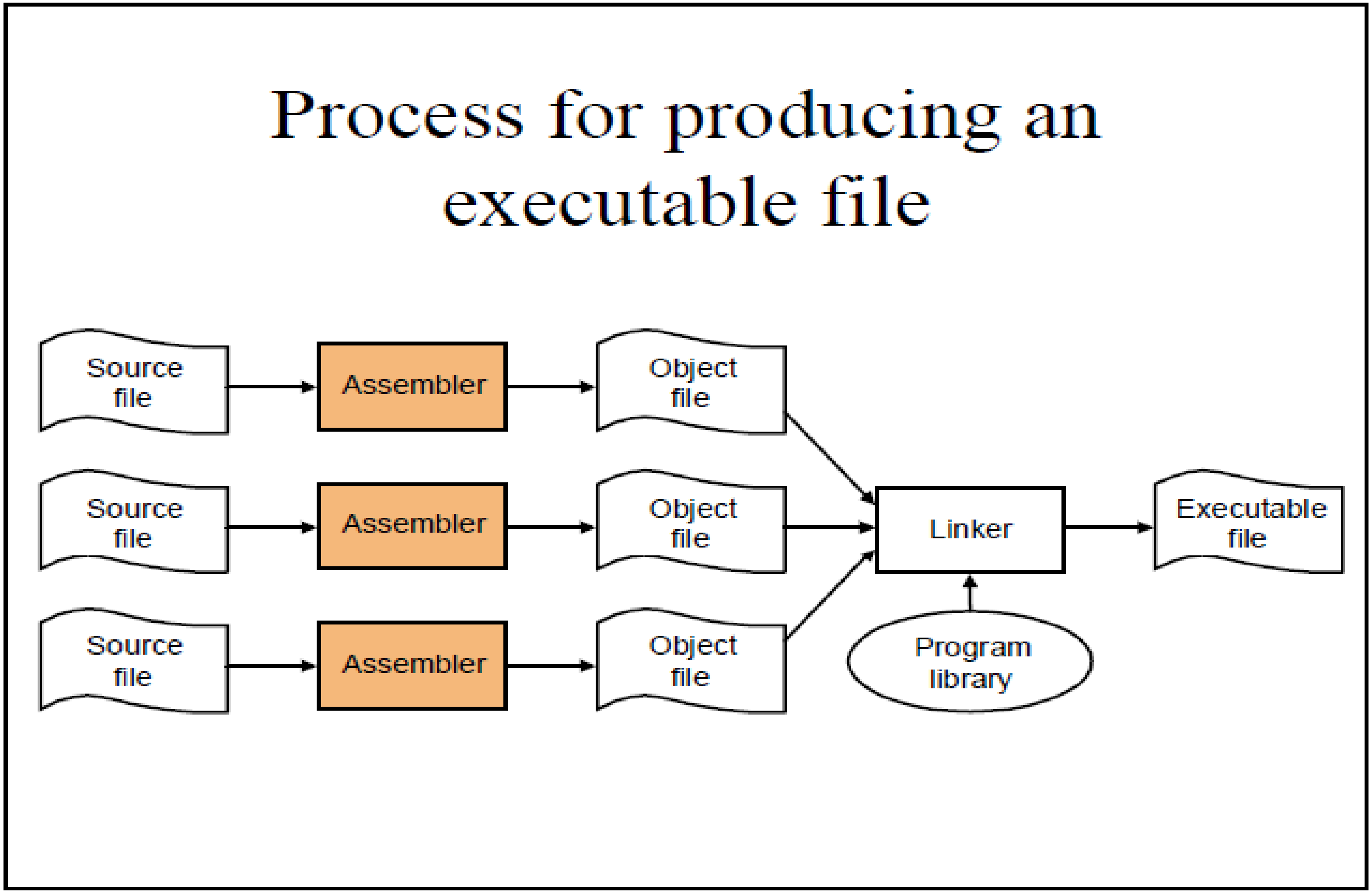
Exercises

Find the output for the following expressions

1. a=a +b *c *2
2. Y= b+c-d+20

Linker and Loader

Linker: A linker, also called link editor or binder, is a program that combines the object modules to form an executable program. In general , in case of a large program, programmers prefer to break the code in to smaller modules, as this simplifies the programming task. Eventually, when the source code of all the modules has been converted in to object code, all the modules need to be put together, which is done by the linker



Process for producing an executable file

Loader

A loader is a special type of a program that copies programs from a storage device to the main memory, where they can be executed.

Front end and Back end of a Compiler

Front end:

I. Lexical Analyzer
II. Syntax Analyzer
III. Semantic Analyzer
IV. Intermediate Code Generator

Back end :

V. Code Optimizer
VI. Code Generator

Advantages of Using Front-end and Back- end

Retargeting: Build a compiler for a new machine by attaching a new code generator to an existing front-end

Optimization: Reuse intermediate code optimizers in compilers for different languages and different machines.

Symbol Table Management

A symbol table is a data structure containing all the identifiers (i.e. names of variables, procedures etc.) of a source program together with all the attributes of each identifier.

For variables, typical attributes include:

- its type,
- how much memory it occupies,
- its scope.

For procedures and functions, typical attributes include:

- the number and type of each argument (if any),
- the method of passing each argument, and
- the type of value returned (if any).

Symbol Table Management

The purpose of the symbol table is to provide quick and uniform access to identifier attributes throughout the compilation process. Information is usually put into the symbol table throughout the analysis phase and used for the synthesis phase.

Error Handler

Each of the six phases (but mainly the analysis phases) of a compiler can encounter errors. On detecting an error the compiler must:

- report the error in a helpful way,
- correct the error if possible, and
- continue processing (if possible) after the error to look for further errors.

Lecture References

A. Aho, R. Sethi and J. Ullman, ***Compilers: Principles, Techniques and Tools*** *(*The Dragon Book*)*, [Second Edition]

References

1. A. Aho, R. Sethi and J. Ullman, ***Compilers: Principles, Techniques and Tools(***The Dragon Book*),* [Second Edition]

2. **Principles of Compiler Design** (2nd Revised Edition 2009) A. A. Puntambekar

3. Basics of Compiler Design Torben Mogensen