3.36pt

Language Processor Introduction 1

Dr. César Ignacio García Osorio
cgosorio@ubu.es

Cao, Thi Huyen(Lilli-L3I2)
caothivananh98@gmail.com

University of Burgos

September 19, 2017

Introduction to Language Processor

```
#gcc hello.c -o hello.exe -fsyntax-only
hello.c: In function `main':
hello.c:3: ‘i’ undeclared
hello.c:8: unterminated string of char
hello.c:9: undefined reference to ‘print’
```

Language Processor

Definition

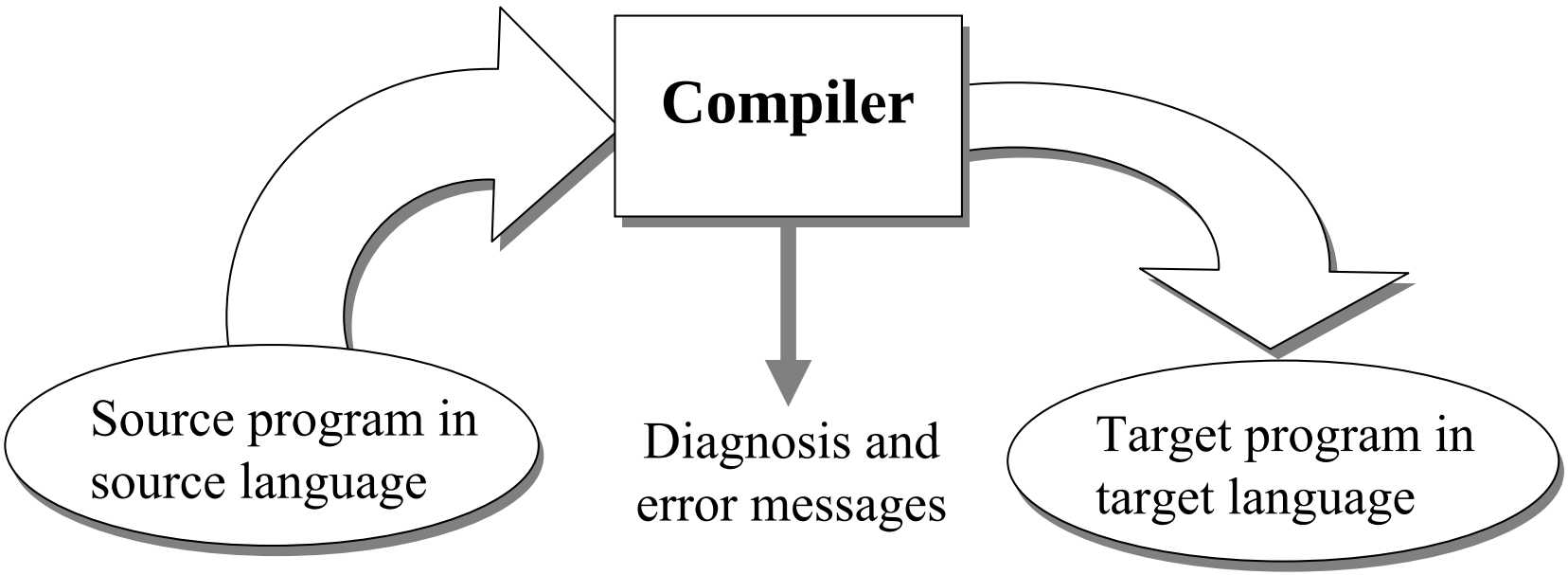
A language processor is a computer program, which takes a sequence of characters (some kind of text in a language) as an input, analyses it, optimizes some processes and returns a result

Emilio Vivancos Rubio, Universidad Politcnica de Valencia
https://www.youtube.com/watch?v=aT5EGLgJs88

Compiler

Definition

A compiler is a program, which translates a program called source program, which is written in a language called source language into a target program of target language. In addition a compiler emits some error messages



Field of use 1

Usage

Compiler building techniques are used in many different fields

- The editor of highlighted syntax makes it easy to edit the program
- The program for text formatting(for example TeX, LaTeX, which from the documentation and formatting commands allows obtaining a suitable file for visualization or printout
- Static syntax checker(such as lint) allows detection of many errors before compilation (something similar to option -**fsyntax-only** of GNU gcc Compiler
- The intepreters (for language like Lisp or Basic, or for programs with their own interpretation languages such as Derive or Mathematics
- Macro languages of extensible applications(Scheme in the world of GNU and Visual Basic for applications in the world of Microsoft Windows)

Field of use 2

- Script languages and "glue" languages (perl, tcl/tk, python) allows rapid development of applications with parts written in different languages
- Analyse of configuration file requires in a lot of modern systems(file .ini of Microsoft Window or source files of X)
- Silicon circuit compiler (for special languages, which describe the layout of printed circuit, obtain the mask for getting the circuit through photographic techniques
- Query languages for databases
- Analyse of protocol specifications for communicating in specified formal languages (such as Abstract Syntax Notation-ASN.1)
- Translation of JSP pages to Servlets Java
- Google Web Toolkit(Java-Javascript translator)
- Translation of style sheet LESS SASS CSS

Field of use 3

- Preprocessor modifies source text in a certain format before compiling (many compilers support macro instructions beyond the language itself, which tells the compiler to include an external file to complete compilation list (Qt, gcc -E)
- Translation of natural language like translation from a natural language to another. For example Spanish to English (There is no achievement currently fundamentally due to the ambiguity of natural language. The greatest achievements in this field is work with a subset of natural language, limit valid syntax construction and/or vacabulary. This topic is usually mentioned in artificial intelligence techniques)
- Bioinformatics Grammars are used to defined languages and more precise grammar are used to model the structure of RNA sequences

A small view of history 1

-
-
-

A small view of history 2

A small view of history 3

A small view of history 4

Use of high level languages

Information

Compiler of high-level languages are currently well established

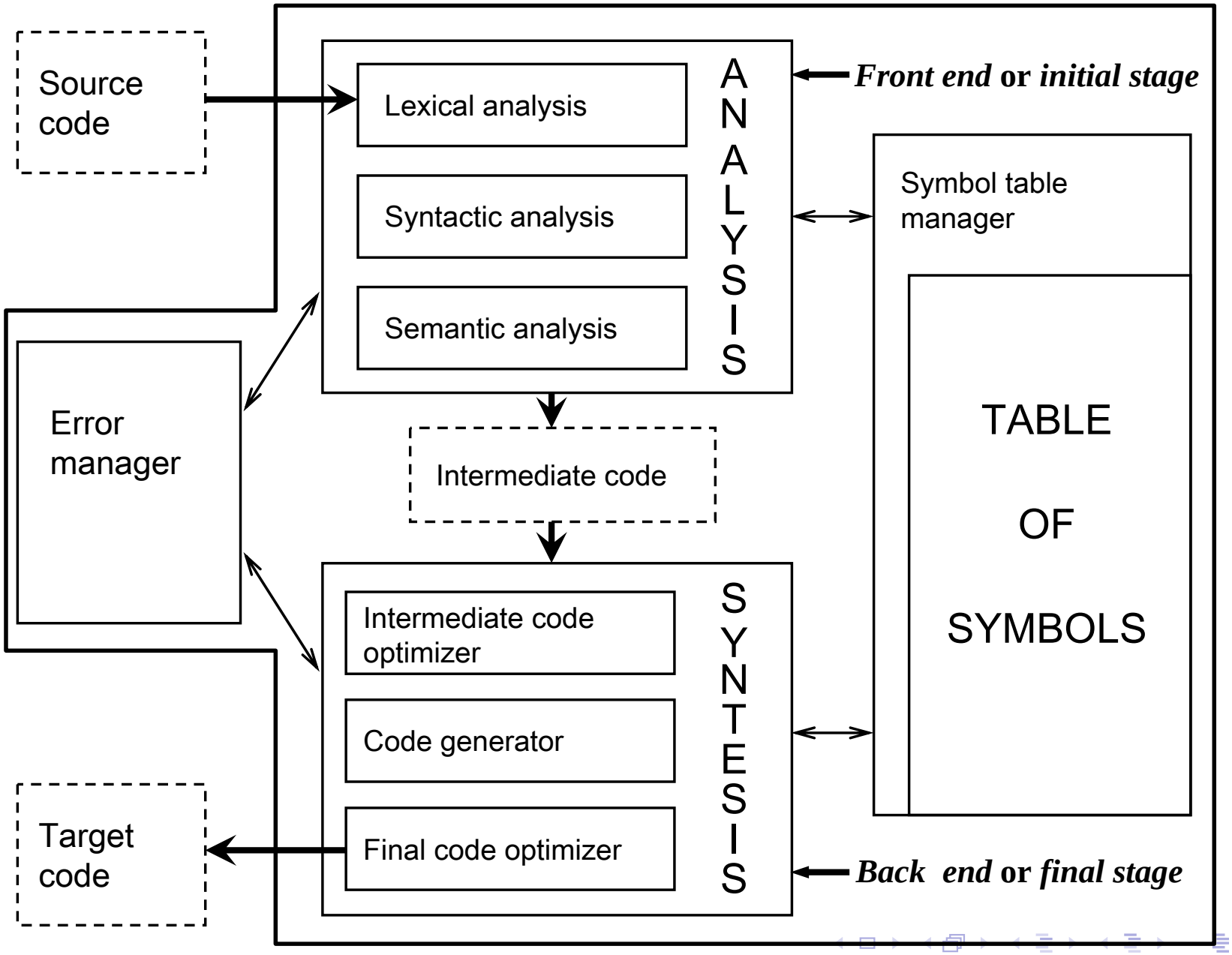
Advantage	Disadvantage
Improving programmer's productivity	Slower than assembler
Reduction of logical errors	big programs
Easy to clean	

Intepreter vs Compiler

- There are two ways to run a program written in high-level languages
  - Compiler: translating the entire program to another equivalent program in machine code. That program is execuatble
  - Interpreter: interpreting program instructions written in high-level language and executing them one by one

Interpreter	Compiler
Easy to locate errors	Dificult to locate errors
Every time the program is executed, its interpretation is necessary	Only one compilation is necessary. After it's done, the execution speed is high
Sufficient in developing and debugging	Sufficient when there is no more error (exploitation)

Structure of a compiler 1



Structure of a compiler 2

The compilation process can be divided into some phases, which simultaneously or consecutively transforms the source program from one represtation to another.

Information

In practice, some phases can be grouped together and temporatory representations between the groups don't need to be built explicitly

front end = lexical + syntax + semantic
(**middle end** = generation of intermediate code
back end = opt. intermediate code + gen. code + opt. code

Structure of a compiler 3

The front-end(analysis or intitial stage) depends mainly on the source language, is designed properly. It's possible to use different back-end(synthesis or final stage) because it includes the phrases, which depend on the target machines. So that the code can be run in different machines

Analyse

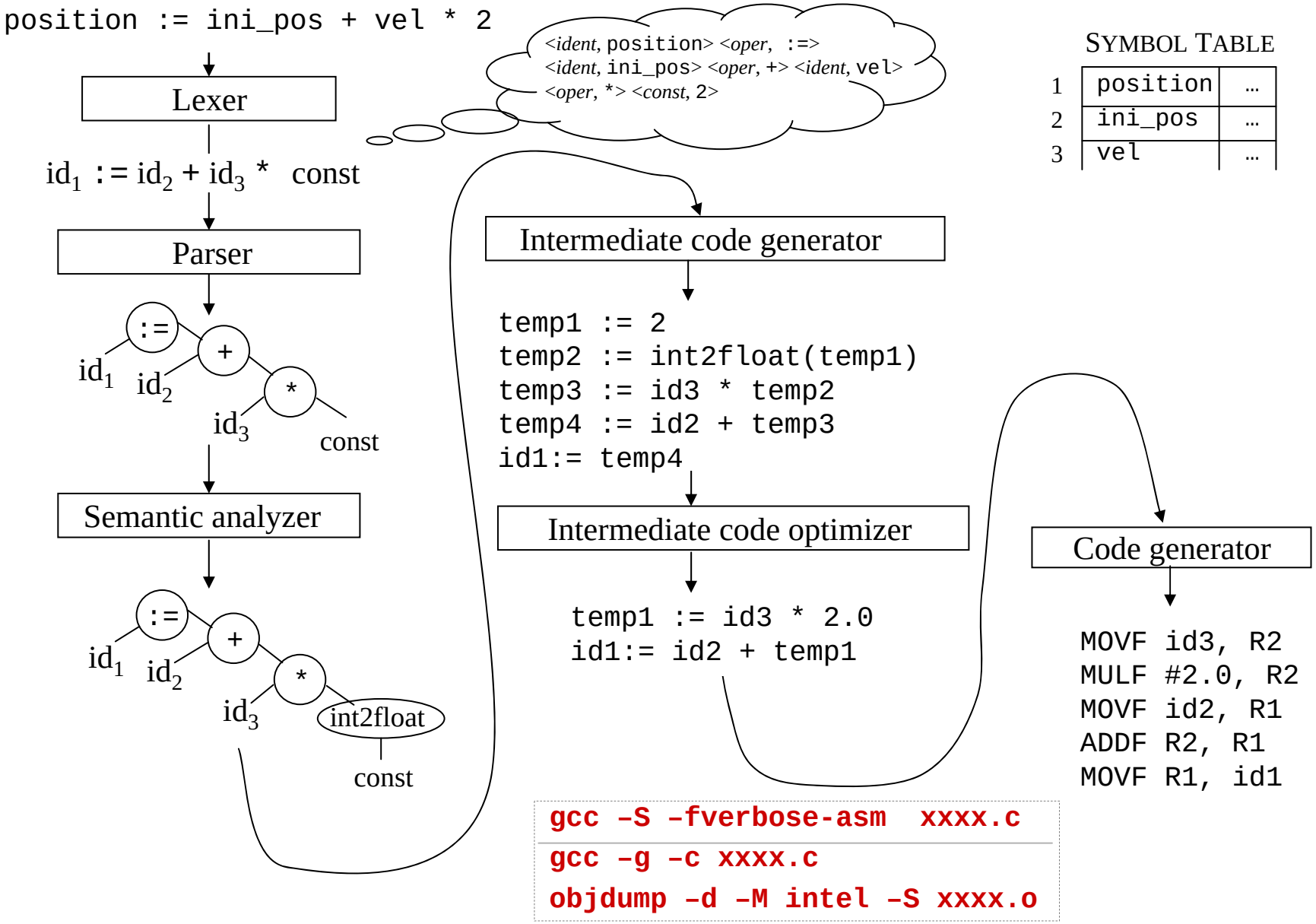
Synthese

Error manager

A compiler has to have a certain behavior against erroneous programs. This process is grouped in a phase called error handler. Each of the previous phases interacts with the error manager

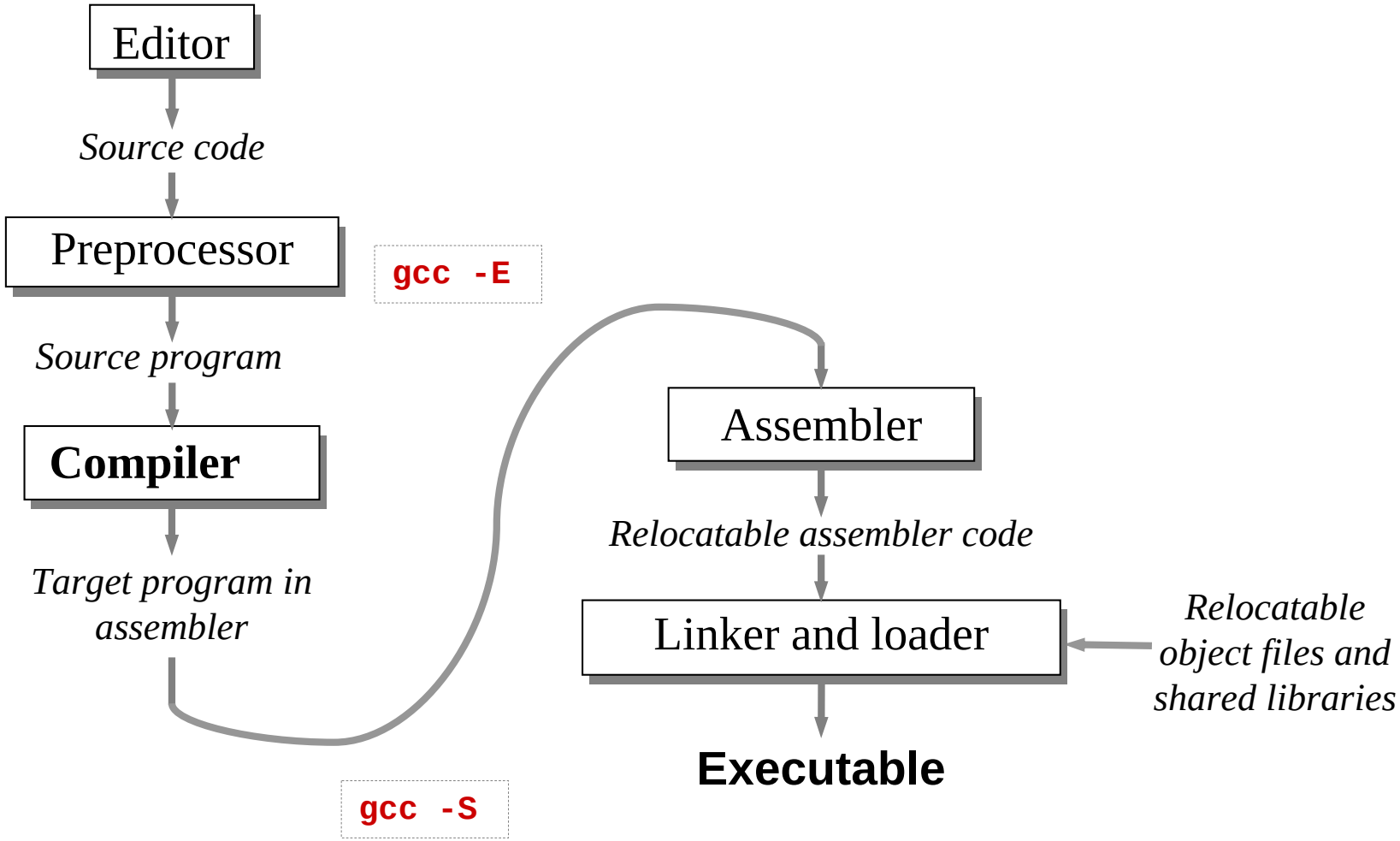
Table of symbols manager

Compiling process 1




```
gcc -S -fverbose-asm  xxxx.c
gcc -g -c xxxx.c
objdump -d -M intel -S xxxx.o
```

Working environment of a compiler



Consruction of a compiler

In building a compiler, there are several approaches, which are:

- in assembler: efficient compilers but difficult to maintain
- in a high-levellanguage: easy to maintain but requires a lot of development time
- using compiler building tools like Flex, Bison, JavaCC, JJTree, JTB

Tools

- Generators of syntax analyzer
  - Generate syntactic analyzers based on context-independent grammar (eg, yacc, bison, JavaCC, ANTLR)
- Generators of lexical analyzers
  - Generate automatically lexical analyzers based on a specification based on regular expressions (eg lex, flex, JavaCC, ANTLR)
- Generators of tree syntax
  - Making it easy to create tree syntax(JJTree, JTB)

Tools

- Syntax-Directed Translation Devices
  - Providing groups of routines, which run through the tree and generating intermediate code
- Automatic code generators
  - Taking a set of rules, which define the translation from intermediate language to machine language. The fundamental technique is matching template
- Devices for data flow analysis
  - Allowing code optimization based on a collection of information about how important that part is while transmitting one to another

Tools

- formal tools
  - Grammar
  - Turing machine
  - derterminate and indeterminate final automat
  - Stack automat
  - Analysis LL, LR, SLR and LALR
- programming tools
  - Yacc/bison
  - Lex/flex/jlex
  - JavaCC, ANTLR, Lemon, Ply, SableCC, CUP...