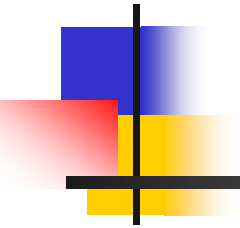
# Compilation and Interpretation



Lecture 06

Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu

# Programming Language Implementation

- A programming language implementation is a system for executing the programs.

- Three general approaches to implementation:
  - ***Compilation***
    - A compiler transforms a program written in a *particular programming language* and turns them into *machine language*
  - ***Interpretation***
    - An interpreter *executes* instructions written in a *particular programming language*
  - ***Hybrid***

# Compilation

- At one extreme, *programs can be translated into machine language*, which can be executed directly on the computer.
- This method is called a ***compiler implementation***
  - Advantage: very fast program execution, once the translation process is complete.
- Most production implementation of languages, such as C, COBOL, and C++, are by compilers.

# COBOL

- "Hello World" example program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO-WORLD.
PROCEDURE DIVISION.
    DISPLAY 'Hello World!'.
STOP RUN.
```

## IDENTIFICATION DIVISION:

- The first mandatory division of every COBOL program. The programmer and the compiler use this division to identify the program.
- PROGRAM-ID specifies the program name that can consist 1 to 30 characters.

# COBOL

- "Hello World" example program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO-WORLD.
PROCEDURE DIVISION.
    DISPLAY 'Hello World!'.
STOP RUN.
```

PROCEDURE DIVISION:

- Used to include the logic of the program.
- Consists of executable statements using variables defined in the data division.
- There must be at least one statement in the procedure division.
- The last statement to end the execution in this division is STOP RUN.

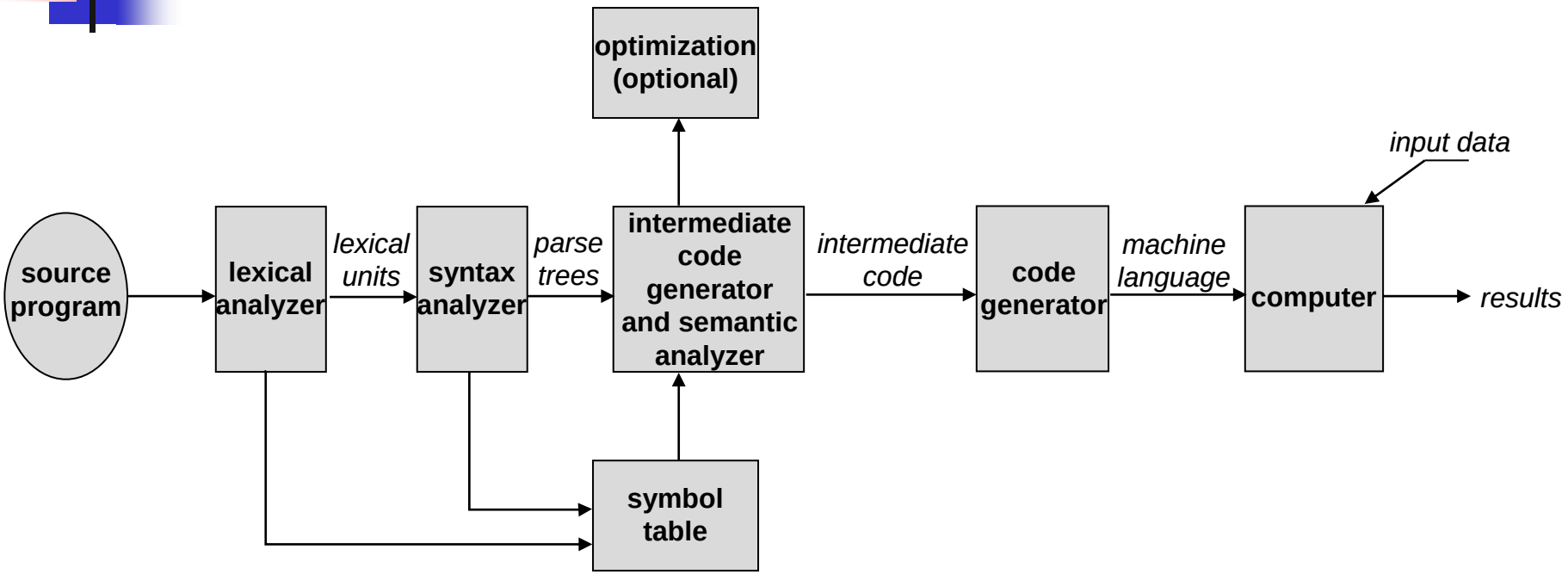
# C

- "Hello World" example program

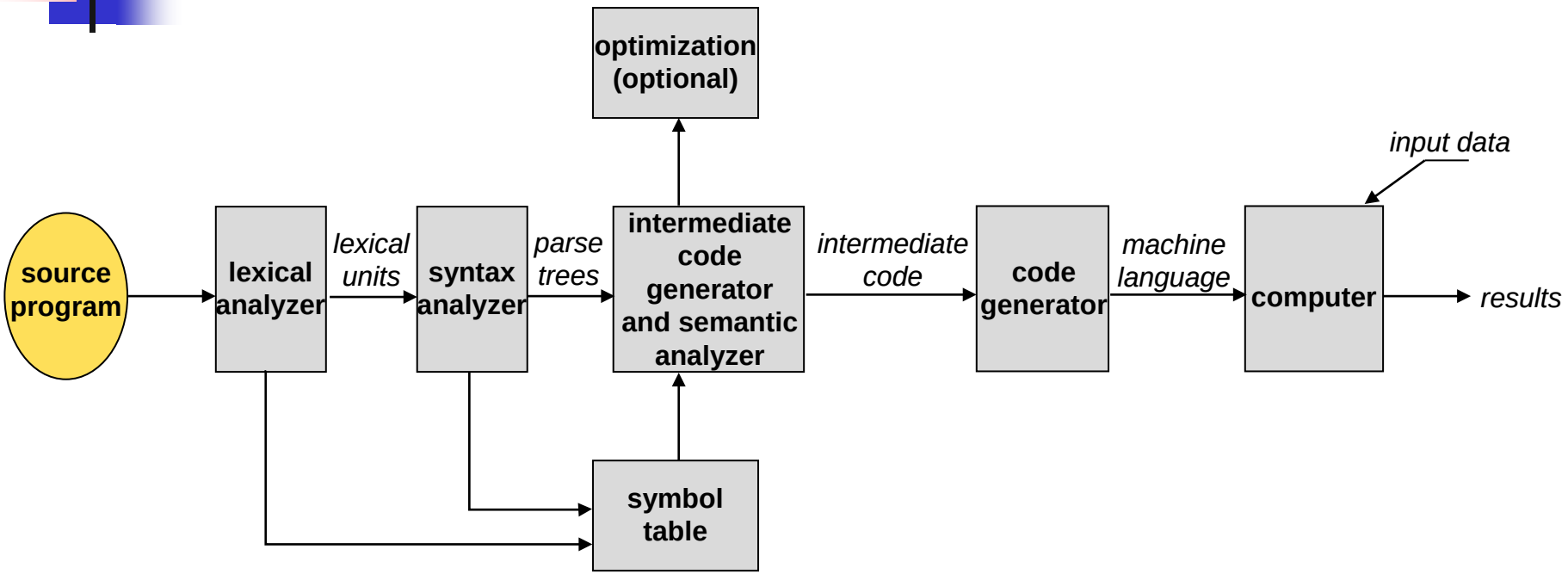
```
#include <stdio.h>
int main() {
    /* printf function displays the content that is
    *  passed between the double quotes.
    */
    printf("Hello World");
    return 0;
}
```

- #include <stdio.h> – This statement tells compiler to include this stdio.h file (library) in the program.
- int main() – Here main() is the function name and *int* is the return type of this function.
- return 0; – As mentioned above, the value 0 means successful execution of main() function.

# Compilation Process



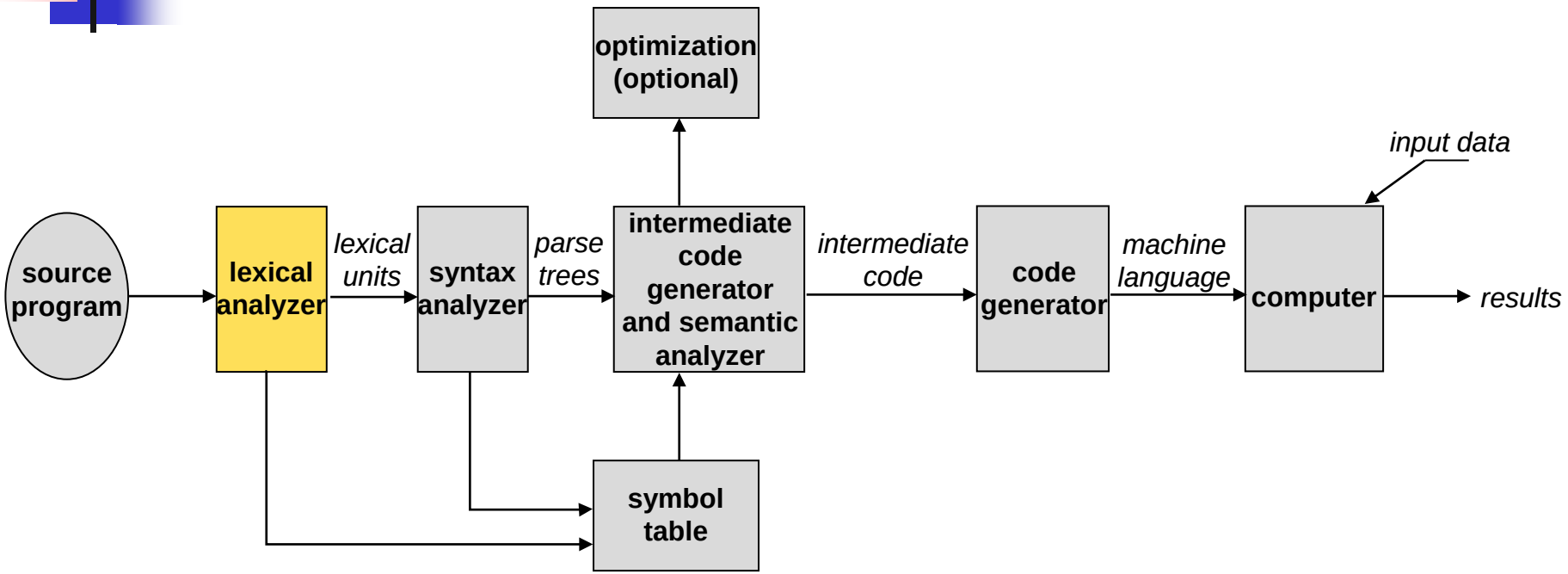
# Compilation Process




**Source language**:

- The language that a compiler translates

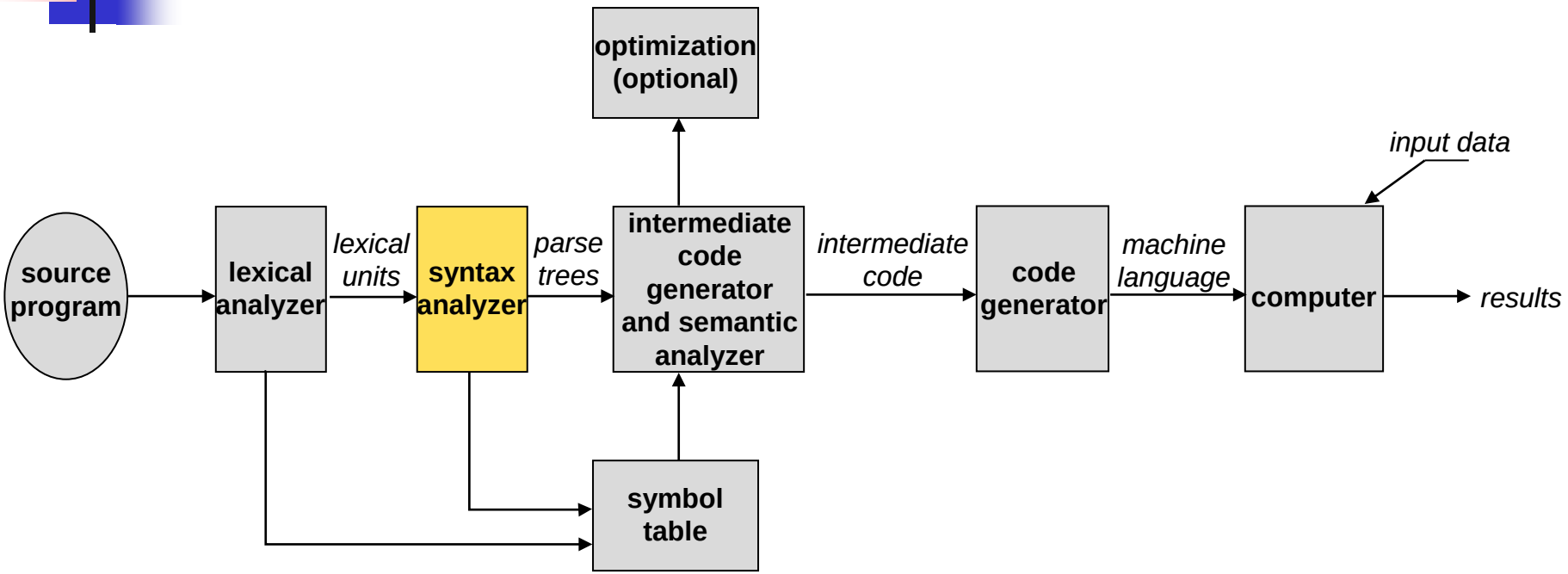
# Compilation Process (cont.)




**Lexical analyzer**:

- Gathers the characters of the source program into *lexical units*
  - *Lexical units* of a program are identifiers, special words, operator, and punctuation symbols
- Ignores comments in the source program

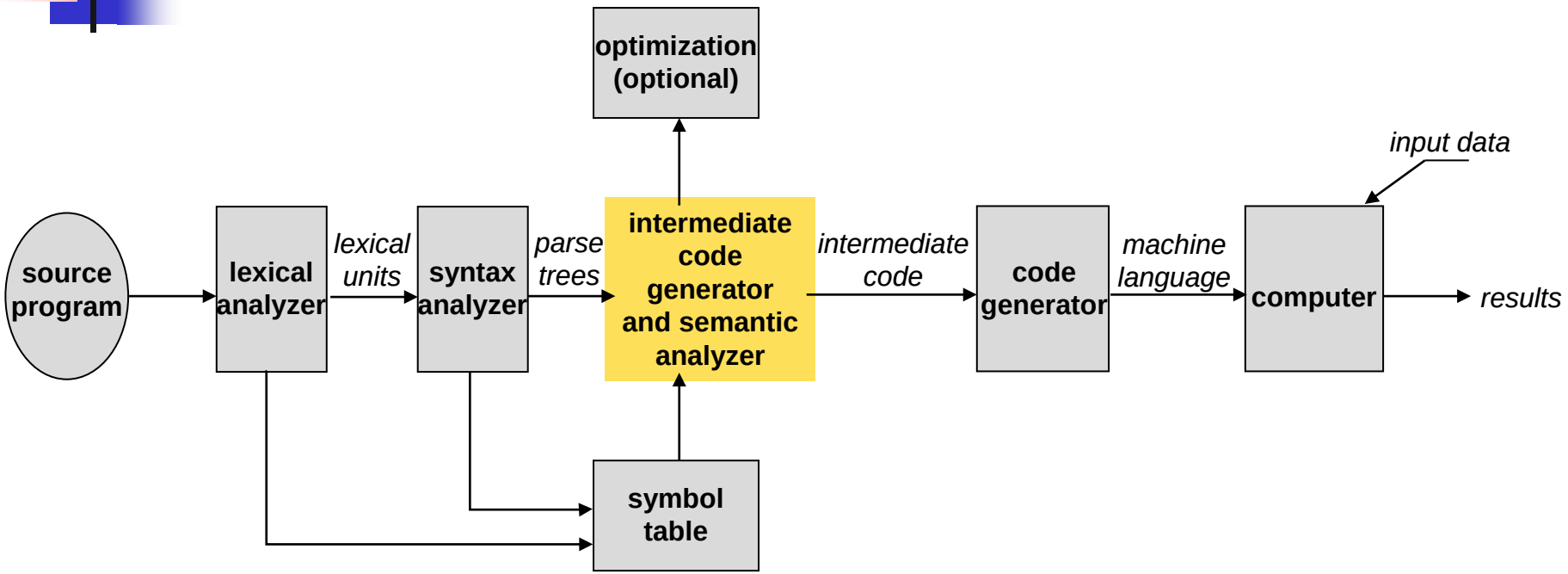
# Compilation Process (cont.)




**Syntax analyzer**:

- Uses lexical units and constructs parse trees
  - *Parse trees* represent the syntactic structure of the program

# Compilation Process (cont.)



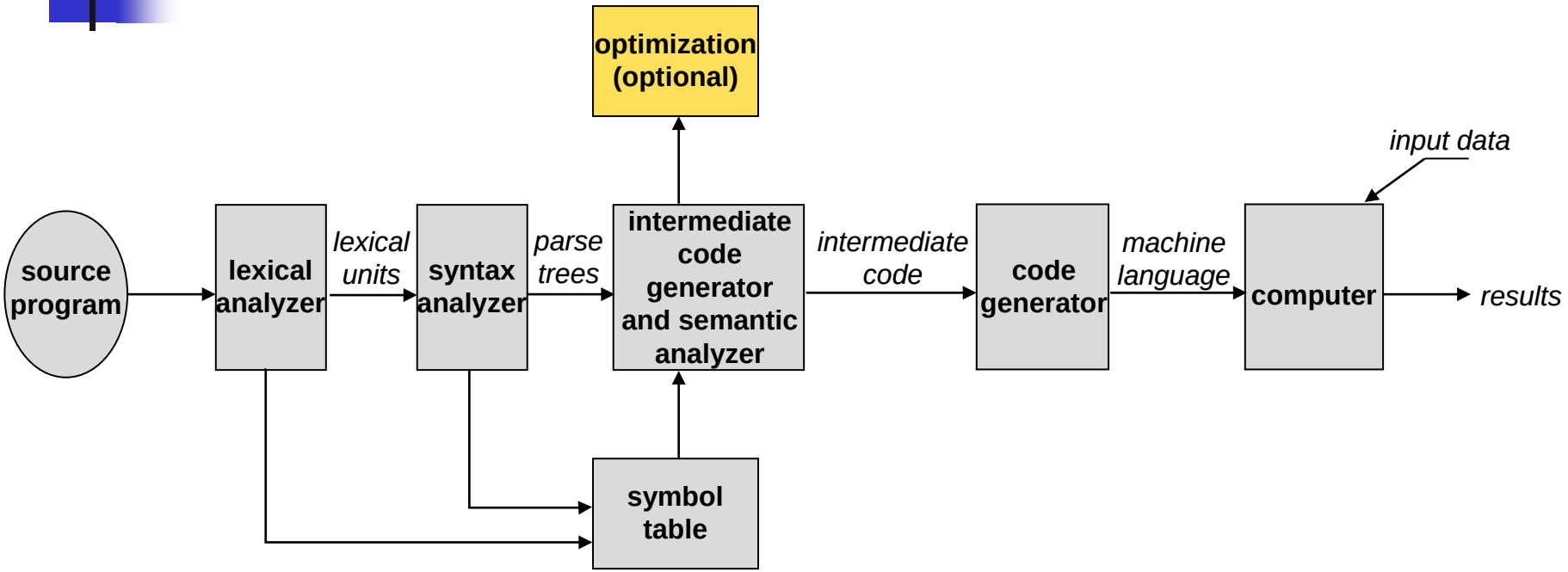

**Intermediate code generator**:

- Produces a program in a different language, which is at intermediate level between the source program and the machine code

**Semantic analyzer**:

- Checks for errors

# Compilation Process (cont.)




**Optimization**:

- Improves programs by making them smaller or faster or both

# Compilation Process (cont.): Optimization

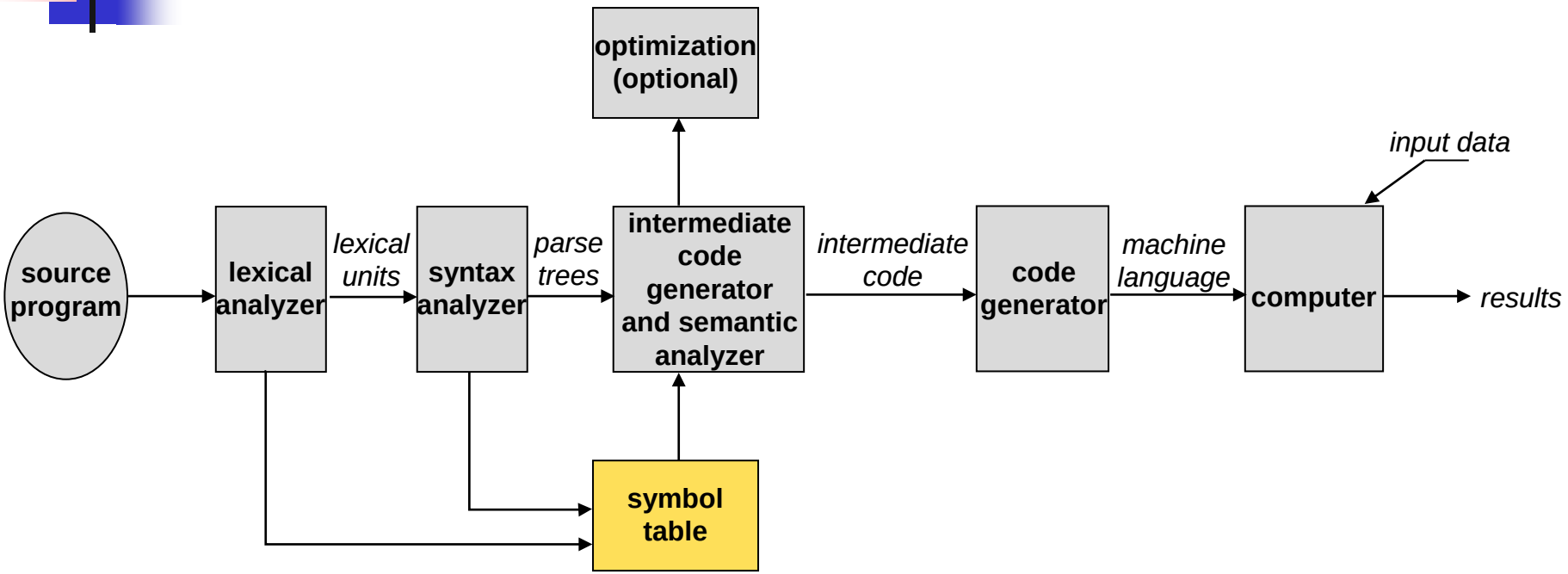
```
do
{
   item = 10;
   value = value + item;
} while (value < 100);
```

This code involves repeated assignment of the identifier item

```
item = 10;
do
{
   value = value + item;
} while (value < 100);
```

This code should save the CPU cycles

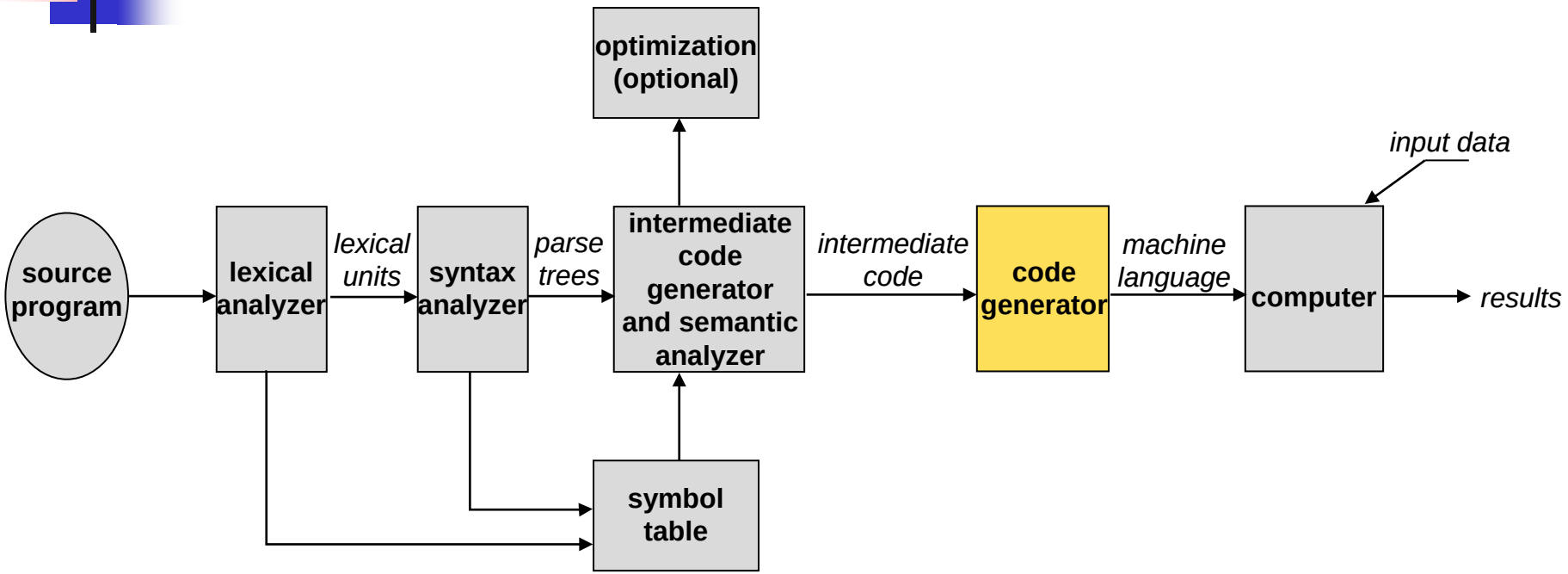
# Compilation Process (cont.)




**Symbol table**:

- Serves as a database for the compilation process
  - Type and attribute information of each user-defined name in the program

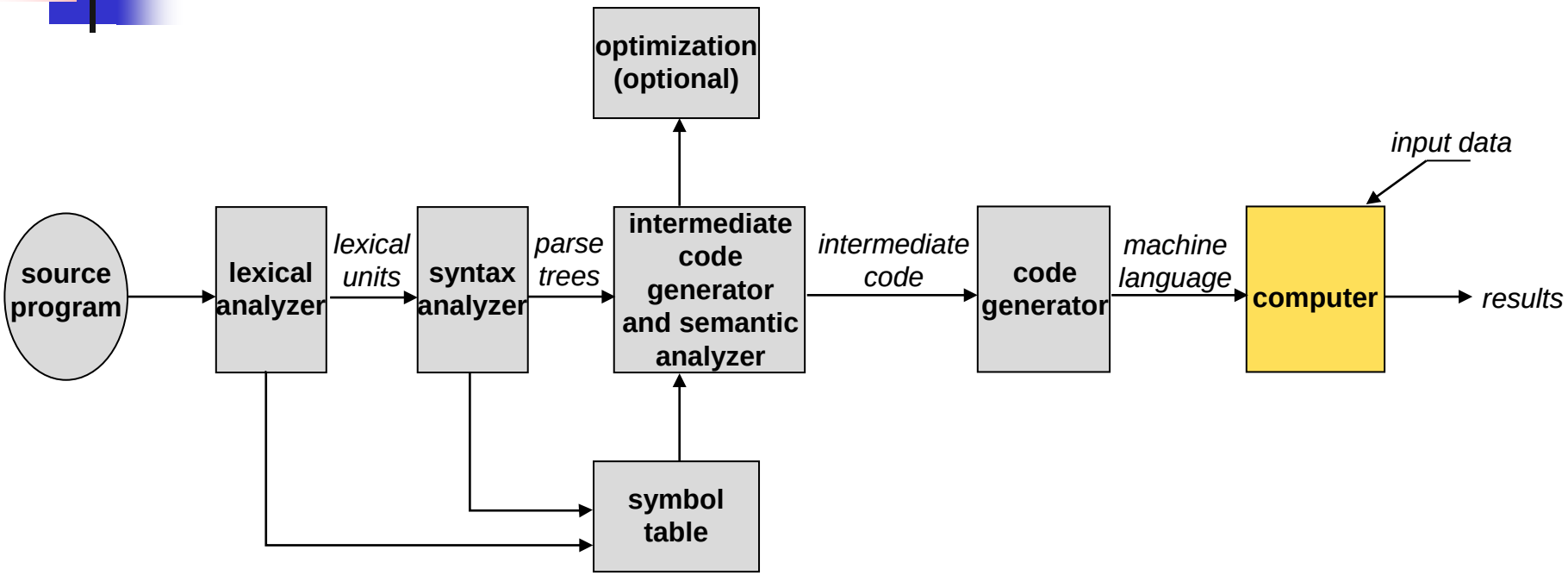
# Compilation Process (cont.)




**Code generator**:

- Translates the optimized intermediate code version of the program into an equivalent machine language program (or machine code)

# Compilation Process (cont.)



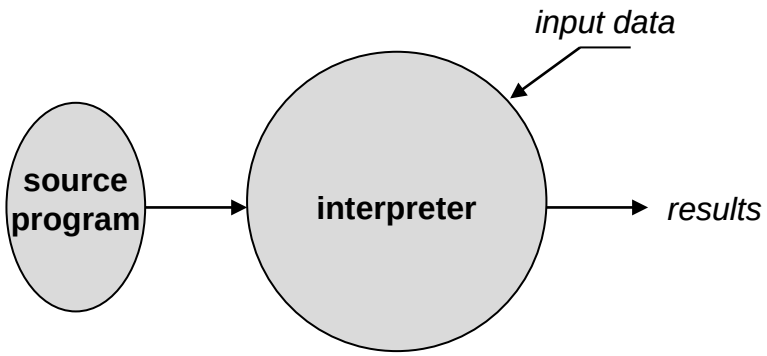

**Computer**:

- Requires programs from the operating system
  - Input and output programs

# Interpretation

- Interpretation
  - Programs are interpreted by another program called an ***interpreter***, with no translation whatever.
  - *Interpreter* acts as a *software simulation of a machine* whose fetch-execute cycle deals with high-level language program statements rather than machine instructions.
- Advantage of interpretation
  - Easy implementation of many source-level debugging operations.
    - For example, array index out of range
- Disadvantages of pure interpretation
  - Execution is 10 to 100 times slower than in compiled systems.
    - The primary source of this slowness is the decoding of the high-level language statements.
    - Regardless of how many times a statements is executed, it must be decoded ever time.

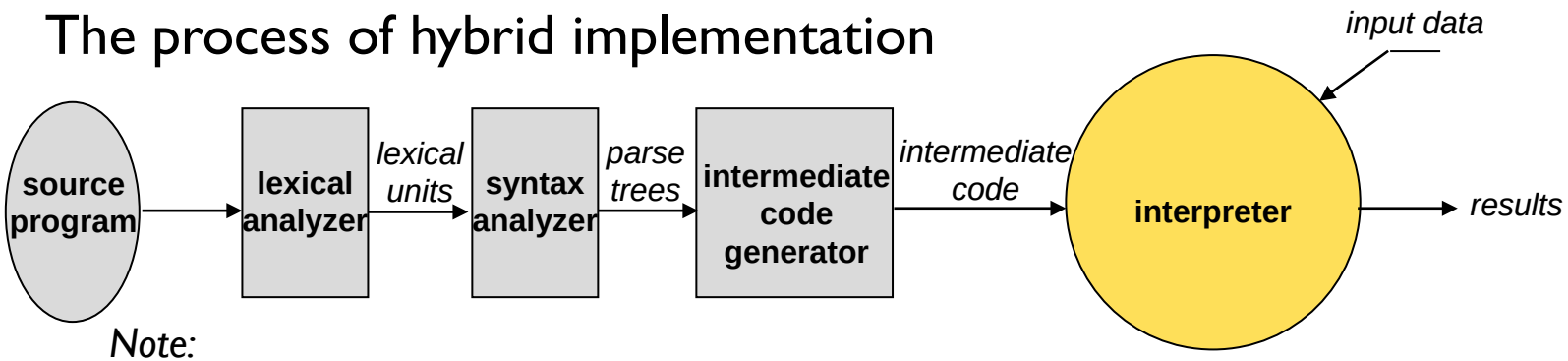
# Interpretation Process



# Hybrid

- Implementation is a compromise between compilers and interpreters
  - Translate high-level language programs to an intermediate language designed to allow easy interpretation.
    - Faster than interpretation because the source language statements are decoded only once.

- The process of hybrid implementation




*Note:*
Instead of translating intermediate language code to machine code, it simply interprets the intermediate code.

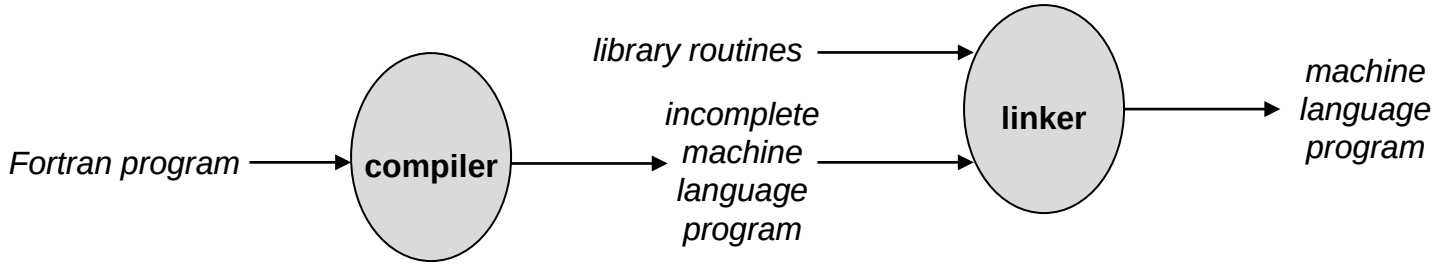
# Hybrid:
# Perl Programming Language

- Perl is implemented with a hybrid system
  - Perl program are partially compiled to detect errors before interpretation and to simplify the interpreter

```
use strict;
use warnings;
print "Hello World\n";
print 23, "\n";
```

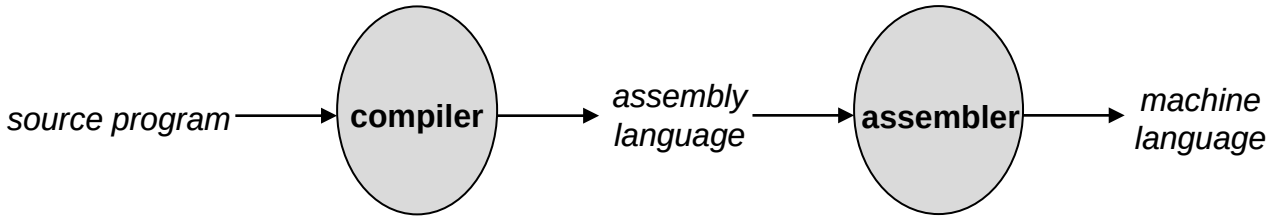
# Implementation Strategies in Practice

- Library of routines and linking
  - Compiler uses a *linker* program to merge the appropriate library of subroutines (e.g., math functions such as sin, cos, etc.) into the final program



# Implementation Strategies in Practice

- Post-compilation Assembly
  - Facilitates debugging
  - Isolates the compiler from changes in the format of machine language files



# Implementation Strategies in Practice

- C ***preprocessor***
  - Removes comments and white space
  - Expands abbreviations in the style of a macro
  - Open file content