CS6109 – COMPILER DESIGN

Module – 3

Presented By
Dr. S. Muthurajkumar,
Assistant Professor,
Dept. of CT, MIT Campus,
Anna University, Chennai.

MODULE - 3

- Error handling
- Error Detection and Recovery
- Lexical phase error management
- Syntax phase error management
- Error recovery routines.

ERROR HANDLING

- The tasks of the Error Handling process are to detect each error, report it to the user, and then make some recovery strategy and implement them to handle the error.
- During this whole process processing time of the program should not be slow.
- Functions of Error Handler:
  - Error Detection
  - Error Report
  - Error Recovery
- Error handler = Error Detection + Error Report + Error Recovery.

ERROR HANDLING

- An Error is the blank entries in the symbol table.
- Errors in the program should be detected and reported by the parser.
- Whenever an error occurs, the parser can handle it and continue to parse the rest of the input.
- Although the parser is mostly responsible for checking for errors, errors may occur at various stages of the compilation process.
- **Types or Sources of Error**
  - Logic errors
  - Run-time error
  - Compile-time errors

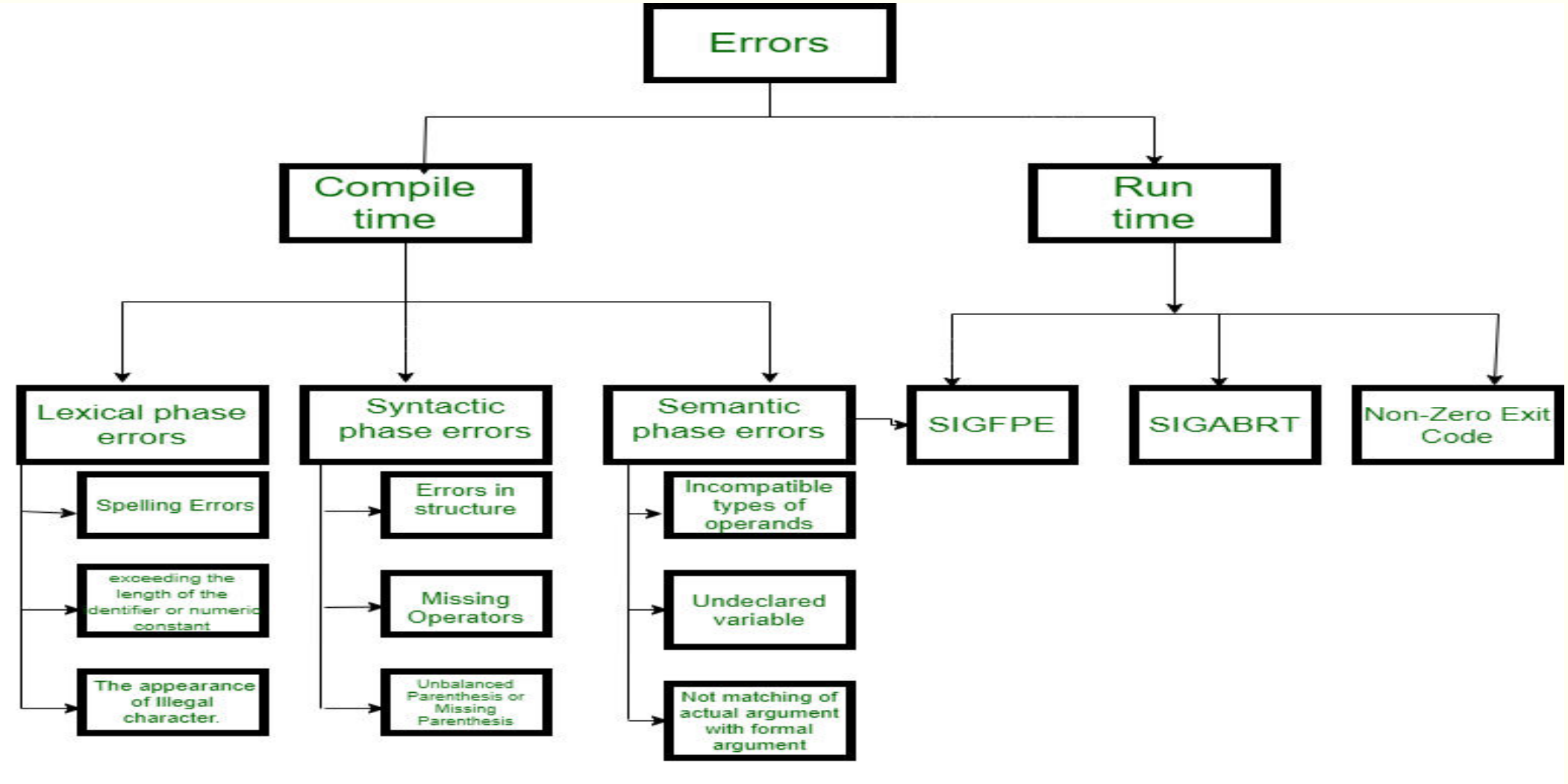
ERROR HANDLING - TYPES

- **Logic errors**
  - Logic errors occur when programs operate incorrectly but do not terminate abnormally (or crash).
  - Unexpected or undesired outputs or other behaviour may result from a logic error, even if it is not immediately recognized as such.
- **Run-time error**
  - A run-time error is an error that takes place during the execution of a program and usually happens because of adverse system parameters or invalid input data.
  - The lack of sufficient memory to run an application or a memory conflict with another program and logical error is an example of this.
  - Logic errors occur when executed code does not produce the expected result.
  - Logic errors are best handled by meticulous program debugging.

ERROR HANDLING - TYPES

- **Compile-time errors**
  - Compile-time errors rise at compile-time, before the execution of the program.
  - Syntax error or missing file reference that prevents the program from successfully compiling is an example of this.

ERROR HANDLING - TYPES



CLASSIFICATION OF COMPILE-TIME ERROR

1. **Lexical:** This includes misspellings of identifiers, keywords or operators
2. **Syntactical:** missing semicolon or unbalanced parenthesis
3. **Semantical:** incompatible value assignment or type mismatches between operator and operand
4. **Logical:** code not reachable, infinite loop.

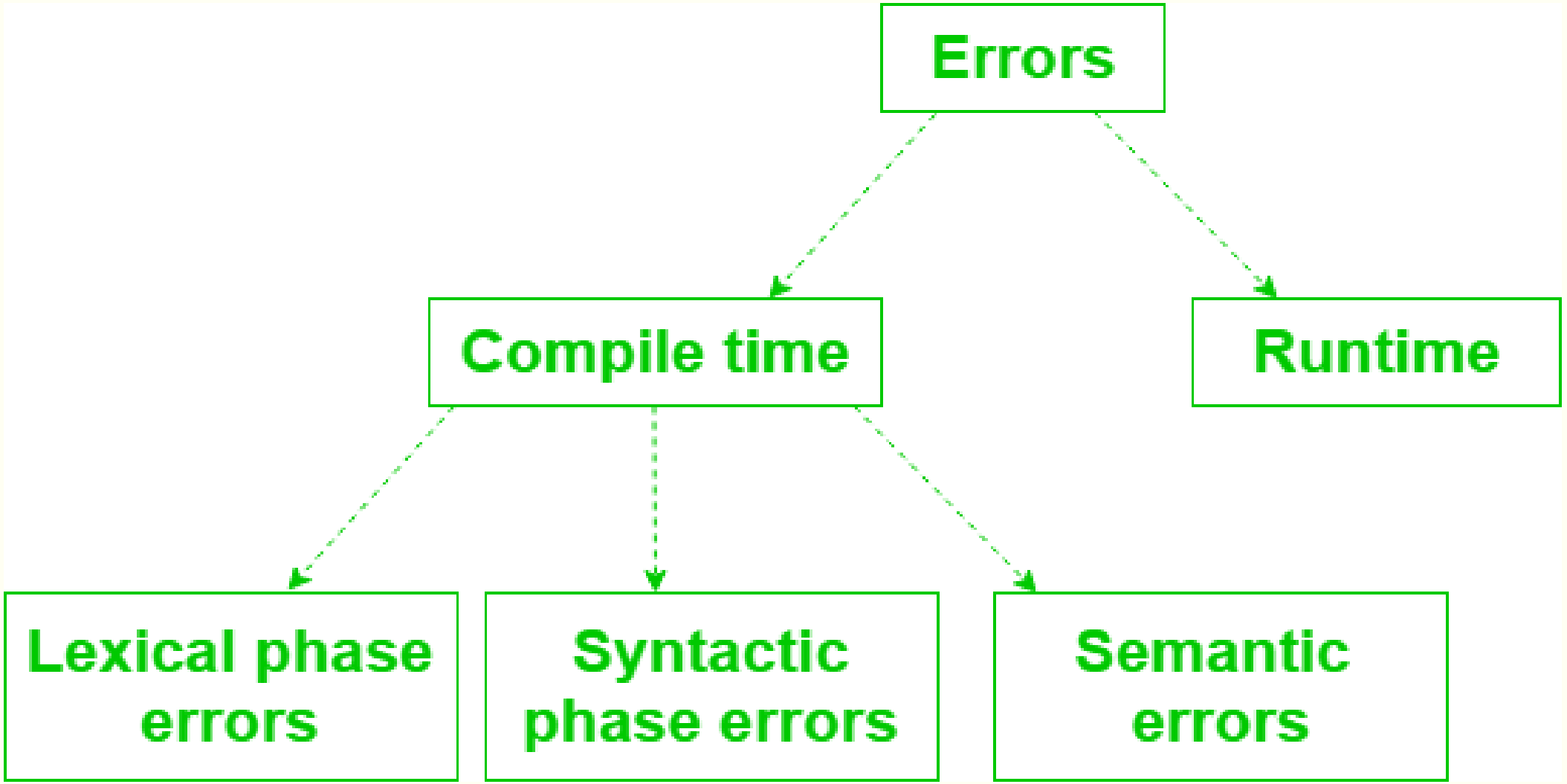
FINDING ERROR OR REPORTING AN ERROR

- **Finding error or reporting an error** – Viable-prefix is the property of a parser which allows early detection of syntax errors.
  - **Goal:** detection of an error as soon as possible without further consuming unnecessary input
  - **How:** detect an error as soon as the prefix of the input does not match a prefix of any string in the language.
  - **Example:** for(;), this will report an error as for have two semicolons inside braces.

ERROR DETECTION AND RECOVERY

- In this phase of compilation, all possible errors made by the user are detected and reported to the user in form of error messages.
- This process of locating errors and reporting it to user is called Error Handling process.
- **Functions of Error handler**
  - Detection
  - Reporting
  - Recovery

ERROR DETECTION AND RECOVERY



ERROR DETECTION AND RECOVERY

- **Lexical phase errors**
- These errors are detected during the lexical analysis phase.
- Typical lexical errors are
  - Exceeding length of identifier or numeric constants.
  - Appearance of illegal characters
  - Unmatched string
- Example 1: **printf(“compiler");$**
- This is a lexical error since an illegal character $ appears at the end of statement.
- Example 2: **This is a comment */**
- This is an lexical error since end of comment is present but beginning is not present.

ERROR DETECTION AND RECOVERY

- **Error recovery:**
- **Panic Mode Recovery**
- In this method, successive characters from the input are removed one at a time until a designated set of synchronizing tokens is found. Synchronizing tokens are delimiters such as; or }
- Advantage is that it is easy to implement and guarantees not to go to infinite loop
- Disadvantage is that a considerable amount of input is skipped without checking it for additional errors

ERROR DETECTION AND RECOVERY

- **Syntactic phase errors**
- These errors are detected during syntax analysis phase.
- Typical syntax errors are
  - Errors in structure
  - Missing operator
  - Misspelled keywords
  - Unbalanced parenthesis
- **Example:** swicth(ch) { .... }
- The keyword switch is incorrectly written as swicth.
- Hence, “Unidentified keyword/identifier” error occurs.

ERROR DETECTION AND RECOVERY

- **Error recovery:**
- **Panic Mode Recovery**
  - In this method, successive characters from input are removed one at a time until a designated set of synchronizing tokens is found. Synchronizing tokens are deli-meters such as ; or }
  - Advantage is that its easy to implement and guarantees not to go to infinite loop
  - Disadvantage is that a considerable amount of input is skipped without checking it for additional errors

ERROR DETECTION AND RECOVERY

- **Error recovery:**
- **Statement Mode recovery**
  - In this method, when a parser encounters an error, it performs necessary correction on remaining input so that the rest of input statement allow the parser to parse ahead.
  - The correction can be deletion of extra semicolons, replacing comma by semicolon or inserting missing semicolon.
  - While performing correction, atmost care should be taken for not going in infinite loop.
  - Disadvantage is that it finds difficult to handle situations where actual error occured before point of detection.

ERROR DETECTION AND RECOVERY

- **Error recovery:**
- **Error production**
  - If user has knowledge of common errors that can be encountered then, these errors can be incorporated by augmenting the grammar with error productions that generate erroneous constructs.
  - If this is used then, during parsing appropriate error messages can be generated and parsing can be continued.
  - Disadvantage is that its difficult to maintain.

ERROR DETECTION AND RECOVERY

- **Error recovery:**
- **Global Correction**
  - The parser examines the whole program and tries to find out the closest match for it which is error free.
  - The closest match program has less number of insertions, deletions and changes of tokens to recover from erroneous input.
  - Due to high time and space complexity, this method is not implemented practically.

ERROR DETECTION AND RECOVERY

- **Semantic errors**
- These errors are detected during semantic analysis phase.
- **Typical semantic errors are**
  - Incompatible type of operands
  - Undeclared variables
  - Not matching of actual arguments with formal one
- **Example:** int a[10], b; .... a = b;
- It generates a semantic error because of an incompatible type of a and b.

ERROR DETECTION AND RECOVERY

- **Error recovery**
- If error “Undeclared Identifier” is encountered then, to recover from this a symbol table entry for corresponding identifier is made.
- If data types of two operands are incompatible then, automatic type conversion is done by the compiler.

ERROR DETECTION AND RECOVERY

- The basic requirement for the compiler is to simply stop and issue a message, and cease compilation.
- There are some common recovery methods that are as follows.
    1. Panic mode recovery
    2. Phase level recovery
    3. Error productions
    4. Global correction

ERROR DETECTION AND RECOVERY

- **Panic mode recovery:**
- This is the easiest way of error-recovery and also, it prevents the parser from developing infinite loops while recovering error.
- The parser discards the input symbol one at a time until one of the designated (like end, semicolon) set of synchronizing tokens (are typically the statement or expression terminators) is found.
- This is adequate when the presence of multiple errors in same statement is rare.
- **Example:** Consider the erroneous expression- (1 + + 2) + 3.
- **Panic-mode recovery:** Skip ahead to next integer and then continue.
- **Bison:** use the special terminal error to describe how much input to skip.
- E->int|E+E|(E)|error int|(error)

PHASE LEVEL RECOVERY

- Perform local correction on the input to repair the error. But error correction is difficult in this strategy.
- When an error is discovered, the parser performs local correction on the remaining input.
- If a parser encounters an error, it makes the necessary corrections on the remaining input so that the parser can continue to parse the rest of the statement.
- You can correct the error by deleting extra semicolons, replacing commas with semicolons, or reintroducing missing semicolons.
- To prevent going in an infinite loop during the correction, utmost care should be taken.
- Whenever any prefix is found in the remaining input, it is replaced with some string. In this way, the parser can continue to operate on its execution.

ERROR DETECTION AND RECOVERY

- **Error productions:**
- Some common errors are known to the compiler designers that may occur in the code.
- Augmented grammars can also be used, as productions that generate erroneous constructs when these errors are encountered.
- The use of the error production method can be incorporated if the user is aware of common mistakes that are encountered in grammar in conjunction with errors that produce erroneous constructs.
- When this is used, error messages can be generated during the parsing process, and the parsing can continue.
- **Example:** write 5x instead of 5*x

ERROR DETECTION AND RECOVERY

- **Global correction:**
- In order to recover from erroneous input, the parser analyzes the whole program and tries to find the closest match for it, which is error-free.
- The closest match is one that does not do many insertions, deletions, and changes of tokens.
- This method is not practical due to its high time and space complexity.

ERROR DETECTION AND RECOVERY

Error / Recovery Method	Lexical Phase Error	Syntactic Phase Error	Semantic Phase Error
Panic Mode	✓	✓	x
Phrase-Level	x	✓	x
Error Production	x	✓	x
Global Production	x	✓	x
Using Symbol Table	x	x	✓

LEXICAL PHASE ERROR MANAGEMENT

- During the lexical analysis phase this type of error can be detected.
- Lexical error is a sequence of characters that does not match the pattern of any token. Lexical phase error is found during the execution of the program.
- **Lexical phase error can be:**
  - Spelling error.
  - Exceeding length of identifier or numeric constants.
  - Appearance of illegal characters.
  - To remove the character that should be present.
  - To replace a character with an incorrect character.
  - Transposition of two characters.

LEXICAL PHASE ERROR MANAGEMENT

Example:

```
Void main()
{
    int x=10, y=20;
    char * a;
    a= &x;
    x= 1xab;
}
```

In this code, 1xab is neither a number nor an identifier. So this code will show the lexical error.

SYNTAX PHASE ERROR MANAGEMENT

- During the syntax analysis phase, this type of error appears.
- Syntax error is found during the execution of the program.
- **Some syntax error can be:**
  - Error in structure
  - Missing operators
  - Unbalanced parenthesis
- When an invalid calculation enters into a calculator then a syntax error can also occurs.
- This can be caused by entering several decimal points in one number or by opening brackets without closing them.

SYNTAX PHASE ERROR MANAGEMENT

For example 1: Using "=" when "==" is needed.

```
16   if (number=200)
17         count << "number is equal to 200";
18   else
19      count << "number is not equal to 200"
```

SYNTAX PHASE ERROR MANAGEMENT

- **The following warning message will be displayed by many compilers:**
- **Syntax Warning:** assignment operator used in if expression line 16 of program firstprog.cpp
- In this code, if expression used the equal sign which is actually an assignment operator not the relational operator which tests for equality.
- Due to the assignment operator, number is set to 200 and the expression number=200 are always true because the expression's value is actually 200. For this example the correct code would be:
- 16 if (number==200)

SYNTAX PHASE ERROR MANAGEMENT

- The following warning message will be displayed by many compilers:

Example 2: Missing semicolon:

```
int a = 5          // semicolon is missing
```

Compiler message:

```
ab.java:20: ';' expected
int a = 5
```

Example 3: Errors in expressions:

```
x = (3 + 5;  // missing closing parenthesis ')'
y = 3 + * 5;   // missing argument between '+' and '*'
```

REFERENCE

1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, “Compilers: Principles, Techniques and Tools”, Second Edition, Pearson Education Limited, 2014.
2. Randy Allen, Ken Kennedy, “Optimizing Compilers for Modern Architectures: A Dependence-based Approach”, Morgan Kaufmann Publishers, 2002.
3. Steven S. Muchnick, “Advanced Compiler Design and Implementation”, Morgan Kaufmann Publishers - Elsevier Science, India, Indian Reprint, 2003.
4. Keith D Cooper and Linda Torczon, “Engineering a Compiler”, Morgan Kaufmann Publishers, Elsevier Science, 2004.
5. V. Raghavan, “Principles of Compiler Design”, Tata McGraw Hill Education Publishers, 2010.
6. Allen I. Holub, “Compiler Design in C”, Prentice-Hall Software Series, 1993.


Thank you