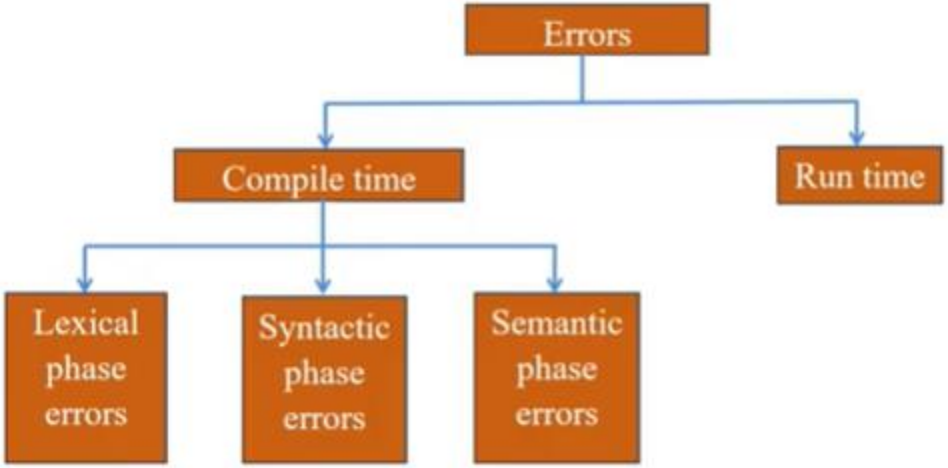
❑ Topics to be covered

- ✓ Types of errors
- ✓ Error recovery strategies

Types of errors

❑ Basic types of errors



❖ Compile time error

1. Lexical error

- ✓ Lexical errors can be detected during lexical analysis phase.
- ✓ Typical lexical phase errors are:
  1. Spelling errors
  2. Exceeding length of identifier or numeric constants
  3. Appearance of illegal characters
- Example:

```
fi ( )
{
}
```

- ✓ In above code 'fi' cannot be recognized as a misspelling of keyword *if* rather lexical analyzer will understand that it is an identifier and will return it as valid identifier.
- ✓ Thus misspelling causes errors in token formation.

2. Syntax error

- ✓ Syntax error appear during syntax analysis phase of compiler.
- ✓ Typical syntax phase errors are:
  1. Errors in structure
  2. Missing operators
  3. Unbalanced parenthesis / missing parenthesis
- ✓ The parser demands for tokens from lexical analyzer and if the tokens do not satisfy the grammatical rules of programming language then the syntactical errors get raised.
- Example:

printf("Hello World !!!") ⟵ **Error: Semicolon missing**

3. Semantic error

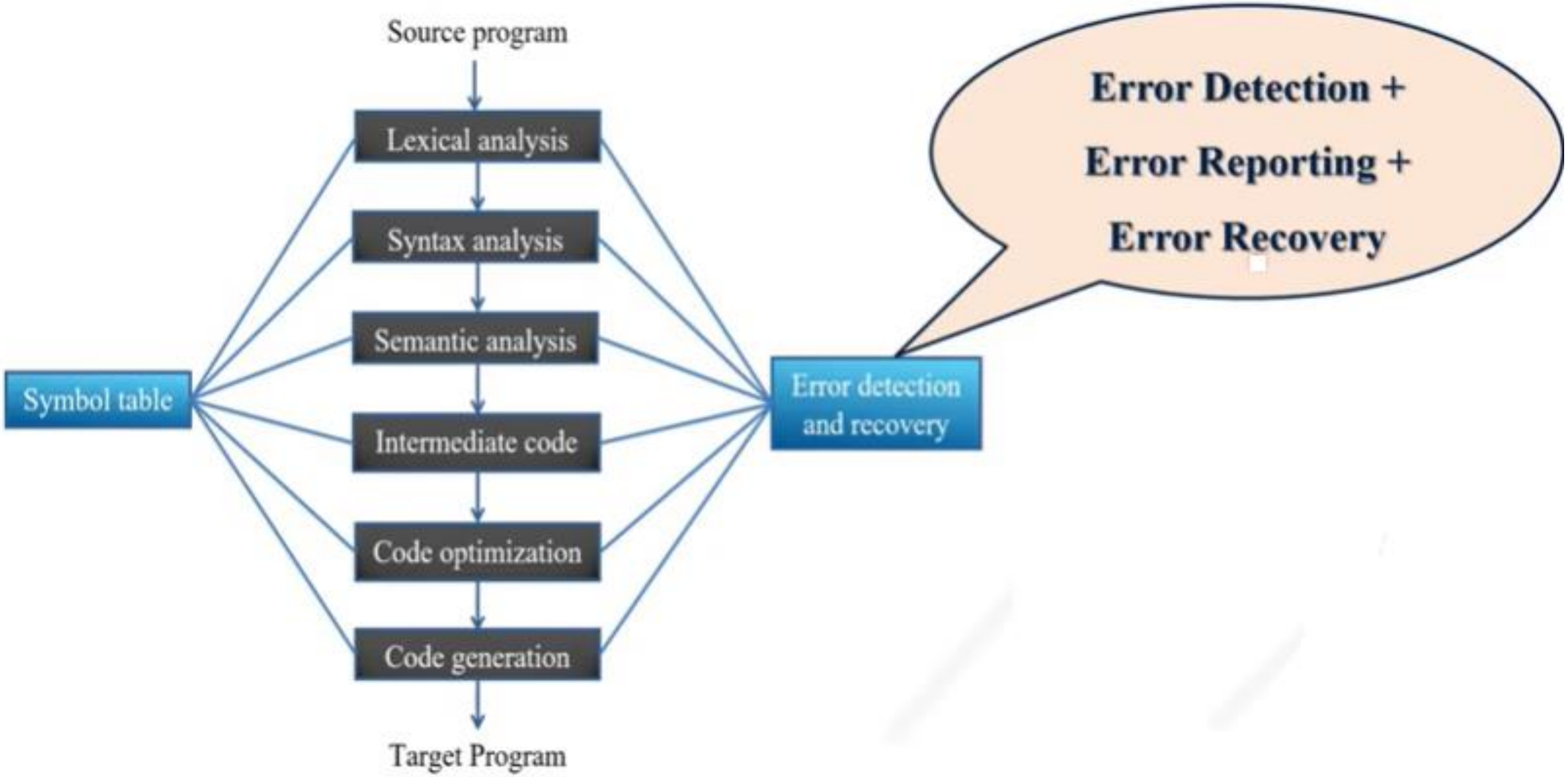
- ✓ Semantic error detected during semantic analysis phase.
- ✓ Typical semantic phase errors are:
  1. Incompatible types of operands
  2. Undeclared variable
  3. Not matching of actual argument with formal argument
- Example:

  id1=id2+id3*60 (Note: id1, id2, id3 are real)

Error recovery strategies

(Ad-Hoc & systematic methods)

❖ Error recovery strategies (Ad-Hoc & systematic methods)



❖ Error recovery strategies (Ad-Hoc & systematic methods)

✓ There are mainly four error recovery strategies:

1. Panic mode
2. Phrase level recovery
3. Error production
4. Global correction

1. Panic mode

- ✓ This strategy is used by most parsing methods. This is simple to implement.
- ✓ In this method on discovering error, the parser discards input symbol one at a time. This process is continued until one of a designated set of synchronizing tokens is found.
- ✓ Synchronizing tokens are delimiters such as semicolon or end. These tokens indicate an end of the input statement.
- ✓ Thus in panic mode recovery a considerable amount of input checking it for additional errors.
- ✓ If there is less number of errors in the same statement then this strategy is best choice.
- **Example:**

int a, 5abcd, sum, $2;

1. Panic mode

- ✓ This strategy is used by most parsing methods. This is simple to implement.
- ✓ In this method on discovering error, the parser discards input symbol one at a time. This process is continued until one of a designated set of synchronizing tokens is found.
- ✓ Synchronizing tokens are delimiters such as semicolon or end. These tokens indicate an end of the input statement.
- ✓ Thus in panic mode recovery a considerable amount of input checking it for additional errors.
- ✓ If there is less number of errors in the same statement then this strategy is best choice.

- **Example:**

int a, ~~5abcd~~, sum, ~~$2~~;

2. Phrase level recovery

- ✓ In this method, on discovering an error parser performs local correction on remaining input.
- ✓ It can replace a prefix of remaining input by some string. This actually helps parser to continue its job.
- ✓ The local correction can be replacing comma by semicolon, deletion of semicolons or inserting missing semicolon. This type of local correction is decided by compiler designer.
- ✓ While doing the replacement a care should be taken for not going in an infinite loop.
- ✓ This method is used in many error-repairing compilers.

3. Error production

- ✓ If we have good knowledge of common errors that might be encountered, then we can augment the grammar for the corresponding language with error productions that generate the erroneous constructs.
- ✓ If error production is used during parsing, we can generate appropriate error message to indicate the erroneous construct that has been recognized in the input.
- ✓ This method is extremely difficult to maintain, because if we change grammar then it becomes necessary to change the corresponding productions.
- For Example: suppose the input string is **abcd**

Grammar:

S→A
A→aA | bA | a | b
B→ cd

Grammar:

$E \rightarrow SB$

$S \rightarrow A$

$A \rightarrow aA \mid bA \mid a \mid b$

$B \rightarrow cd$

4. Global correction

- ✓ We often want such a compiler that makes very few changes in processing an incorrect input string.
- ✓ Given an incorrect input string x and grammar G, the algorithm will find a parse tree for a related string y, such that number of insertions, deletions and changes of token require to transform x into y is as small as possible.
- ✓ Such methods increase time and space requirements at parsing time.
- ✓ Global production is thus simply a theoretical concept.

Thank You