Error Detection and Recovery

Errors

- The compiler should be able to detect errors in the source program
- To some extent it should also be able to recover from these errors
- A simple compiler may stop all activities other than lexical and syntactic analysis after the detection of the first error.
- A more complex compiler may attempt to repair the error and transform the erroneous input into a similar but legal input on which normal processing can be resumed.

Reporting errors

- Good error diagnostics can help reduce debugging and maintenance effort
  - The message should pinpoint the errors in terms of original source rather than internal representation
  - “missing right parenthesis in line 5” rather than “error code 412”
  - The error message should not be redundant.

Sources of errors

- Errors can be classified based on how they are introduced
  - Algorithmic errors
  - Coding errors
  - Keypunching or transcription error
  - exceeding the machine limit
  - Compiler errors -compiler may insert errors during translation

Syntactic errors

- Examples of syntactic errors
  - Missing right parenthesis
    - Sum=((a+b)/2
  - Extraneous comma
  - : in place of ;
  - misspelled keyword
  - extra blank
    - /* comments * /

Syntactic errors

- Classification of errors
  1. Deletion error –Missing right parenthesis

  2. Insertion error -Extraneous comma, extra blank
  3. Replacement error- : in place of ;

  4. Transposition error-misspelled keyword

Minimum distance correction of syntactic errors

- A theoretical way of defining errors and their location is the minimum Hamming distance method
- A program **P** has ***k*** errors if the shortest sequence of error transformations that will map any valid program into **P** has length ***k***

Minimum distance correction of syntactic errors

- If our error transformations be the insertion of a single character or the deletion of a single character, then the program fragment (a) is distance one from (b)

(a)

ifa = b then
sum=sum+a;
else
sum=sum-a;

(b)

If a = b then
sum=sum+a;
else
sum=sum-a;

- *Fragment (a) can be mapped into (b) by the insertion of a blank between if and a.*
- *Fragment (b) is a minimum distance correction of (a)*

Semantic errors

- Detected at compile and run time
- Most common are errors of declaration and scope
  - undeclared or multiple declared identifiers
  - incompatibilities between **operators** and **operand**, **formal** parameters and **actual** parameters
  - **e,g strongly typed languages are easier**

Dynamic errors

- Some languages have certain kind of errors that can only be detected at run time
  - E.g.,
    - Range checking for arrays,
    - or a jump to some unknown memory location

Lexical Phase errors

- no prefix of the remaining input fits the specification of any token class
- it can invoke an error recovery routine
  – simplest is to skip erroneous characters until the lexical analyzer can find another token

Syntactic Phase errors

- Most of the error detection and recovery in a compiler is centered around syntax analysis
- A parser detects an error when it has no legal move from its current configuration
- To recover from an error a parser should ideally
  - locate the position of the error
  - Correct the error
  - Revise its current configuration,
  - and resume parsing
- *But there is no guarantee that the error has been successfully corrected*

Syntactic Phase errors

- Time of detection
  - LL (I) and LR(I) parsers have the valid prefix property
    - Announce error as soon as a prefix of the input has been seen for which there is no valid continuation
- Panic mode
  - discards input symbols until a “synchronizing” token usually a statement delimiter is encountered
  - deletes stack entries until it finds an entry such that it can continue parsing

Error Recovery in Operator Precedence parsing

- Two points in the parsing process at which an operator-precedence parser can discover syntactic errors
  1. If no precedence relation holds between the terminal on top of the stack and the current input symbol
  2. if a handle has been found but there is no production with this handle at the right side
- Error detection and error recovery routine can be divided into several pieces.
  - One piece handles errors of type 2.e.g., this routine might pop symbols off the stack
  - A diagnostic message is printed

	+	*	(	)	id	$
+	·>	<·	<·	·>	<·	·>
*	·>	·>	<·	·>	<·	·>
(	<·	<·	<·	$\doteq$	<·	e1
)	·>	·>	e2	·>	e2	·>
id	·>	·>	e2	·>	e2	·>
$	<·	<·	<·	e3	<·	e4

1.3. Operator precedence matrix with error entries.

```
e1: /* called when expression ends with a left parenthesis */
    pop ( from the stack
    issue diagnostic:  ILLEGAL LEFT PARENTHESIS

e2: /* called when id or ) is followed by id or ( */
    insert + onto the input
    issue diagnostic:  MISSING OPERATOR

e3: /* called when expression begins with a right parenthesis */
    delete ) from the input
    issue diagnostic:  ILLEGAL RIGHT PARENTHESIS

e4: /* called when expression is null */
    insert id onto the input
    issue diagnostic:  MISSING EXPRESSION
```

Error Recovery LR Parsing and LL parsing

- An LR parser will detect an error when it examine each error entry in the parsing table
  - an appropriate recovery procedure can be constructed
- Another way is to scan down the stack until a state s with a goto on a particular nonterminal A is found.
  - and discard zero or more symbols until a symbol *a* is found that can follow A and resumes normal parsing

Error recovery from semantic errors

- Recovery from an undeclared name
  - Make an entry of this name in the symbol table with appropriate attributes
  - The attribute should be in the context in which it is used
    - E.g., a name is use as a real or as a procedure name
  - Each time it is incorrectly used it is check from the symbol table whether it has been entered and a message was given earlier,
    - so no need to repeat the same error message again