Error Handling

Error handling refers to the response and recovery procedures from error conditions present in a software application. In other words, it is the process comprised of anticipation, detection and resolution of application errors, programming errors or communication errors. Error handling helps in maintaining the normal flow of program execution. In fact, many applications face numerous design challenges when considering error-handling techniques.

Error handling helps in handling both hardware and software errors gracefully and helps execution to resume when interrupted. When it comes to error handling in software, either the programmer develops the necessary codes to handle errors or makes use of software tools to handle the errors. In cases where errors cannot be classified, error handling is usually done with returning special error codes. Special applications known as error handlers are available for certain applications to help in error handling. These applications can anticipate errors, thereby helping in recovering without actual termination of application.

There are four main categories of errors:

* Logical errors
* Generated errors
* Compile-time errors
* Runtime errors

Error-handling techniques for development errors include rigorous proofreading. Error-handling techniques for logic errors or bugs is usually by meticulous application debugging or troubleshooting. Error-handling applications can resolve runtime errors or have their impact minimized by adopting reasonable countermeasures depending on the environment. Most hardware applications include an error-handling mechanism which allows them to recover gracefully from unexpected errors.

As errors could be fatal, error handling is one of the crucial areas for application designers and developers, regardless of the application developed or programming languages used. In worst-case scenarios, the error handling mechanisms force the application to log the user off and shut down the system.

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:

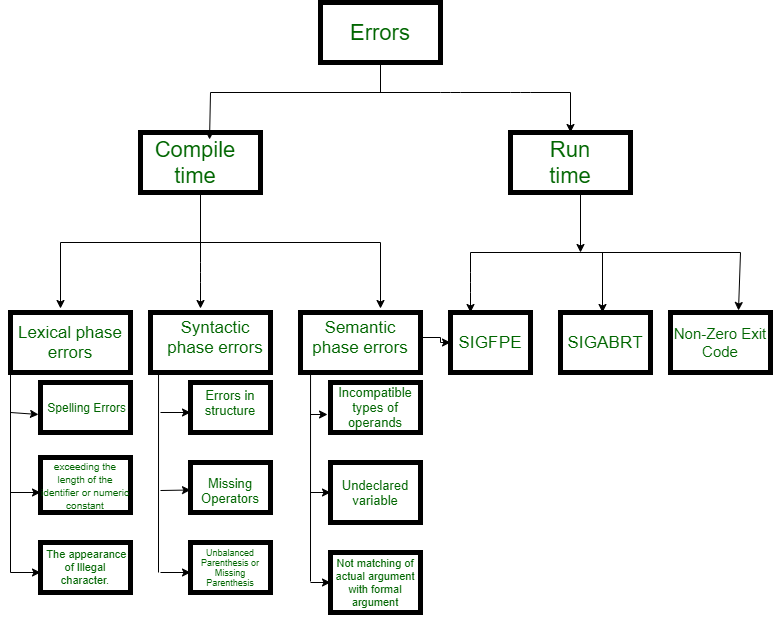
* Error Detection
* Error Report
* Error Recovery

Error handler=Error Detection+Error Report+Error Recovery.

 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.  
So, there are many types of errors and some of these are:  
**Types**or**Sources of Error –** There are three types of error: logic, run-time and compile-time error: 

1. **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.
2. 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.
3. **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.



**Classification of Compile-time error –**

1. **Lexical**: This includes misspellings of identifiers, keywords or operators
2. **Syntactical**: a 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 –** Viable-prefix is the property of a parser that 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 having two semicolons inside braces.

**Error 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.

We already discuss the errors. Now, let’s try to understand the recovery of errors in every phase of the compiler.

**1. 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 the same statement is rare. Example: Consider the erroneous expression- (1 + + 2) + 3. Panic-mode recovery: Skip ahead to the 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)

**2. Phase level recovery:**  
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.

**3. Error productions:**  
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 

**4. 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.