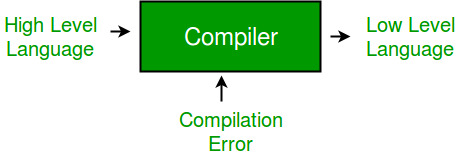
**Experiment No. 1: Introduction to Compilers**

A **translator** or language processor is a program that translates an input program written in a programming language into an equivalent program in another language. The **compiler** is a type of translator that takes a program written in a high-level programming language as input and translates it into an equivalent program in low-level languages such as machine language or assembly language.



The program written in a high-level language is known as a **source program**, and the program converted into a low-level language is known as an **object (or target) program**. Without compilation, no program written in a high-level language can be executed. For every programming language, there is a different compiler, but the basic tasks performed by each compiler are the same.

The process of translating source code into machine code involves several stages, including **lexical analysis**, **syntax analysis**, **semantic analysis**, **code generation**, and **optimization**. The compiler is an intelligent program compared to an assembler. It verifies all types of limits, ranges, errors, etc. A compiler program takes more time to run and occupies a large amount of memory space. The speed of the compiler is slower than other system software because it processes the entire program and then performs the translation.

* A **Cross Compiler** runs on machine 'A' and produces code for another machine 'B'. It is capable of creating code for a platform other than the one on which the compiler is running.
* A **Source-to-Source Compiler** (transcompiler or transpiler) is a compiler that translates source code written in one programming language into the source code of another programming language.

**High-Level Programming Language**  
A high-level programming language is a language that abstracts the attributes of a computer. It is more convenient for the user to write programs in high-level languages.

**Low-Level Programming Language**  
A low-level programming language is a language that is closer to machine language and does not require complex programming ideas and concepts.

### Stages of Compiler Design

A diagram of a language

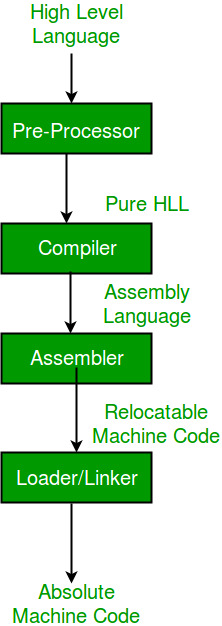
Description automatically generated

1. **Lexical Analysis**  
   The first stage of compiler design is **lexical analysis**, also known as scanning. In this stage, the compiler reads the source code character by character and breaks it down into a series of tokens, such as keywords, identifiers, and operators. These tokens are passed to the next stage of the compilation process.
2. **Syntax Analysis**  
   The second stage is **syntax analysis** or parsing. The compiler checks the syntax of the source code to ensure that it conforms to the rules of the programming language. A **parse tree**, a hierarchical representation of the program's structure, is built to check for syntax errors.
3. **Semantic Analysis**  
   In **semantic analysis**, the compiler checks the meaning of the source code to ensure it makes sense. It performs type checking to ensure variables are used correctly and operations are performed on compatible data types. Other checks include undeclared variables and incorrect function calls.
4. **Code Generation**  
   **Code generation** is the stage where the compiler translates the parse tree into machine code that can be executed by the computer. The code must be efficient and optimized for the target platform.
5. **Optimization**  
   The final stage is **optimization**, where the compiler analyzes the generated code and makes improvements to enhance its performance. Optimizations may include constant folding, loop unrolling, and function inlining.

### Language Processing Systems

A computer is a logical assembly of software and hardware. The hardware understands a language that is difficult for humans to grasp, so we write programs in high-level languages that are easier to understand and maintain. These programs go through a series of transformations so they can be used by machines.

* **High-Level Language**: Programs containing pre-processor directives like #include or #define are written in high-level languages. These directives instruct the pre-processor on how to process the code.
* **Pre-Processor**: The pre-processor removes all #include directives and #define directives. It performs tasks such as file inclusion and macro expansion.
* **Assembly Language**: It is an intermediate state, combining machine instructions with other necessary data for execution.



### Interpreter vs. Compiler

An **interpreter** converts high-level language into machine language like a compiler, but they differ in how they process input. A compiler reads the entire program, translates it, and then executes it. An interpreter, on the other hand, translates the program line by line. Interpreted programs tend to be slower compared to compiled ones.

### Types of Compilers

1. **Single-Pass Compiler**  
   In a single-pass compiler, all the phases of compilation are present inside a single module that converts source code to machine code in one go.
2. **Two-Pass Compiler**  
   A two-pass compiler translates the program twice—once from the front end and then from the back end.
3. **Multi-Pass Compiler**  
   A multi-pass compiler creates several intermediate codes and processes the syntax tree multiple times, breaking the code into smaller components for more efficient translation.

### Phases of a Compiler

The compilation process is divided into two major phases:

1. **Analysis Phase**  
   In this phase, an intermediate representation is created from the source code. It includes:
   * **Lexical Analyzer**
   * **Syntax Analyzer**
   * **Semantic Analyzer**
   * **Intermediate Code Generator**
2. **Synthesis Phase**  
   In this phase, the equivalent target program is created from the intermediate representation. It includes:
   * **Code Optimizer**: Optimizes the abstract code.
   * **Code Generator**: Translates abstract intermediate code into specific machine instructions.

### Compiler Operations

Some of the key operations of a compiler include:

* Breaking the source program into smaller parts.
* Enabling the creation of symbol tables and intermediate representations.
* Detecting errors during the code compilation process.

Converting source code into machine code.