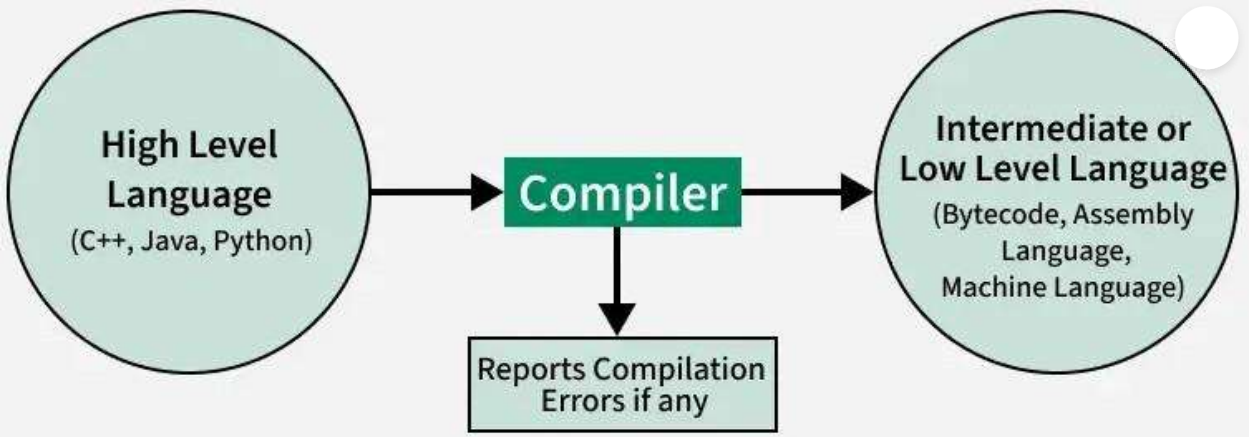
Introduction of Compiler Design

Last Updated : 25 Jan, 2025

A compiler is software that translates or converts a program written in a high-level language (Source Language) into a low-level language (Machine Language or Assembly Language). Compiler design is the process of developing a compiler.



The development of compilers is closely tied to the evolution of programming languages and computer science itself. Here's an overview of how compilers came into existence:

History of Compilers

In the 1950s, Grace Hopper developed the first compiler, leading to languages like FORTRAN (1957), LISP (1958), and COBOL (1959). The 1960s saw innovations like ALGOL, and the 1970s introduced C and

revolutionized programming, enabling complex systems and improving software efficiency.

Please refer History of Compilers for more details.

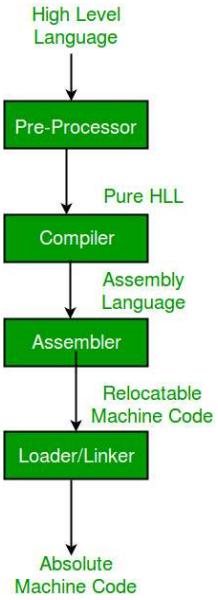
Role of Compilers

A program written in a high-level language cannot run without compilation. Each programming language has its own compiler, but the fundamental tasks performed by all compilers remain the same. Translating source code into machine code involves multiple stages, such as lexical analysis, syntax analysis, semantic analysis, code generation, and optimization.

While compilers are specialized, they differ from general translators. A translator or language processor is a tool that converts an input program written in one programming language into an equivalent program in another language.

Language Processing Systems

We know a computer is a logical assembly of Software and Hardware. The hardware knows a language, that is hard for us to grasp, consequently, we tend to write programs in a high-level language, that is much less complicated for us to comprehend and maintain in our thoughts. Now, these programs go through a series of transformations so that they can readily be used by machines. This is where language procedure systems come in handy.



High-Level Language to Machine Code

- **High-Level Language:** If a program contains pre-processor directives such as #include or #define it is called HLL. They are closer to humans but far from machines. These (#) tags are called preprocessor directives. They direct the pre-processor about what to do.

- **Pre-Processor:** The pre-processor removes all the #include directives by including the files called file inclusion and all the #define directives using macro expansion. It performs file inclusion, augmentation, macro-processing, etc. For example: Let in the source program, it is written #include “Stdio. h”. Pre-Processor replaces this file with its contents in the produced output.

- **Assembly Language:** It’s neither in binary form nor high level. It is an intermediate state that is a combination of machine instructions and some other useful data needed for execution.

- **Assembler:** For every platform (Hardware + OS) we will have an assembler. They are not universal since for each platform we have one. The output of the assembler is called an object file. Its translates assembly language to machine code.

- **Compiler:** The compiler is an intelligent program as compared to an assembler. The compiler verifies all types of limits, ranges, errors, etc. Compiler program takes more time to run and it occupies a huge amount of memory space. The speed of the compiler is slower than other system software. It takes time because it enters through the program and then does the translation of the full program.

- **Interpreter:** An interpreter converts high-level language into low-level machine language, just like a compiler. But they are different in the way they read the input. The Compiler in one go reads the inputs, does the processing, and executes the source code whereas the interpreter does the same line by line. A compiler scans the entire program and translates it as a whole into machine code whereas an interpreter translates the program one statement at a time. Interpreted programs are usually slower concerning compiled ones.

- **Relocatable Machine Code:** It can be loaded at any point and can be run. The address within the program will be in such a way that it will cooperate with the program movement.

- **Loader/Linker:** Loader/Linker converts the relocatable code into absolute code and tries to run the program resulting in a running program or an error message (or sometimes both can happen). Linker loads a variety of object files into a single file to make it executable. Then loader loads it in memory and executes it.
    - **Linker:** The basic work of a linker is to merge object codes (that have not even been connected), produced by the compiler, assembler, standard library function, and operating system resources.
    - **Loader:** The codes generated by the compiler, assembler, and linker are generally re-located by their nature, which means to

Thus the basic task of loaders to find/calculate the exact address of these memory locations.

Overall, compiler design is a complex process that involves multiple stages and requires a deep understanding of both the programming language and the target platform. A well-designed compiler can greatly improve the efficiency and performance of software programs, making them more useful and valuable for users.

Phases of a Compiler

There are two major phases of compilation, which in turn have many parts. Each of them takes input from the output of the previous level and works in a coordinated way.

Analysis Phase

An intermediate representation is created from the given source code :

- Lexical Analyzer
- Syntax Analyzer
- Semantic Analyzer
- Intermediate Code Generator

Synthesis Phase

An equivalent target program is created from the intermediate representation. It has two parts :

- Code Optimizer
- Code Generator

Read more about Phases of Compiler, Here.

Compiler Construction Tools

1. **Parser Generators:** It creates syntax analyzers (parsers) based on grammatical descriptions of programming languages.
2. **Scanner Generators:** It produces lexical analyzers using regular expressions to define the tokens of a language.
3. **Syntax-Directed Translation Engines:** It generates intermediate code in three-address format from input comprising a parse tree.
4. **Automatic Code Generators:** It converts intermediate language into machine language using template matching techniques.
5. **Data-Flow Analysis Engines:** It supports code optimization by analyzing the flow of values throughout different parts of the program.
6. **Compiler Construction Toolkits:** It provides integrated routines to facilitate the construction of various compiler components.

Read more about Compiler Construction Tools here.

Types of Compiler

- **Self Compiler:** When the compiler runs on the same machine and produces machine code for the same machine on which it is running then it is called as self compiler or resident compiler.

- **Cross Compiler**: The compiler may run on one machine and produce the machine codes for other computers then in that case it is called a cross-compiler. It is capable of creating code for a platform other than the one on which the compiler is running.

- **Source-to-Source Compiler:** A Source-to-Source Compiler or transcompiler or transpiler is a compiler that translates source code written in one programming language into the source code of another programming language.

performs the work of converting source code to machine code.

- **Two Pass Compiler:** Two-pass compiler is a compiler in which the program is translated twice, once from the front end and the back from the back end known as Two Pass Compiler.

- **Multi-Pass Compiler:** When several intermediate codes are created in a program and a syntax tree is processed many times, it is called Multi-Pass Compiler. It breaks codes into smaller programs.

- **Just-in-Time (JIT) Compiler: I**t is a type of compiler that converts code into machine language during program execution, rather than before it runs. It combines the benefits of interpretation (real-time execution) and traditional compilation (faster execution).

- **Ahead-of-Time (AOT) Compiler:** It converts the entire source code into machine code before the program runs. This means the code is fully compiled during development, resulting in faster startup times and better performance at runtime.

- **Incremental Compiler:** It compiles only the parts of the code that have changed, rather than recompiling the entire program. This makes the compilation process faster and more efficient, especially during development.

Operations of Compiler

These are some operations that are done by the compiler.

- It breaks source programs into smaller parts.
- It enables the creation of symbol tables and intermediate representations.
- It helps in code compilation and error detection.

Advantages of Compiler Design

- **Efficiency:** Compiled programs are generally more efficient than interpreted programs because the machine code produced by the compiler is optimized for the specific hardware platform on which it will run.
- **Portability:** Once a program is compiled, the resulting machine code can be run on any computer or device that has the appropriate hardware and operating system, making it highly portable.
- **Error Checking:** Compilers perform comprehensive error checking during the compilation process, which can help catch syntax, semantic, and logical errors in the code before it is run.
- **Optimizations:** Compilers can make various optimizations to the generated machine code, such as eliminating redundant instructions or rearranging code for better performance.

Disadvantages of Compiler Design

- **Longer Development Time:** Developing a compiler is a complex and time-consuming process that requires a deep understanding of both the programming language and the target hardware platform.
- **Debugging Difficulties:** Debugging compiled code can be more difficult than debugging interpreted code because the generated machine code may not be easy to read or understand.
- **Lack of Interactivity:** Compiled programs are typically less interactive than interpreted programs because they must be compiled before they can be run, which can slow down the development and testing process.
- **Platform-Specific Code:** If the compiler is designed to generate machine code for a specific hardware platform, the resulting code may not be portable to other platforms.

GATE CS Corner Questions

- GATE CS 2011, Question 1
- GATE CS 2011, Question 19
- GATE CS 2009, Question 17
- GATE CS 1998, Question 27
- GATE CS 2008, Question 85
- GATE CS 1997, Question 8
- GATE CS 2014 (Set 3), Question 65
- GATE CS 2015 (Set 2), Question 29

Frequently Asked Questions on Compiler Design – FAQs

What is a compiler?

A compiler is a tool that converts high-level code into machine code.

Why is it so important?

It ensures that programs are executed correctly and run in an efficient manner.

What are the major stages?

It breaks up the code, checks, optimizes, and generates machine code.

How does a compiler differ from an interpreter?

A compiler translates the entire program in one go; whereas an interpreter does this line by line.

Does the compiler optimize code?

Yes, it can do so during compilation.

Comment | More info | Advertise with us

Next Article

Compiler construction tools

Similar Reads

Introduction To Compilers

A Compiler is a software that typically takes a high level language (Like C++ and Java) code as input and converts the input to a lower level language at...

15+ min read

Compiler Design Tutorial

A compiler is software that translates or converts a program written in a high-level language (Source Language) into a low-level language (Machine...

Code optimization is a crucial phase in compiler design aimed at enhancing the performance and efficiency of the executable code. By improving the...

15+ min read

Phases of a Compiler

A compiler is a software tool that converts high-level programming code into machine code that a computer can understand and execute. It acts as a...

15+ min read

Intermediate Code Generation in Compiler Design

In the analysis-synthesis model of a compiler, the front end of a compiler translates a source program into an independent intermediate code, then th...

15+ min read

Introduction to Syntax Analysis in Compiler Design

Syntax Analysis (also known as parsing) is the step after Lexical Analysis. The Lexical analysis breaks source code into tokens. Tokens are inputs for...

15+ min read

Incremental Compiler in Compiler Design

Incremental Compiler is a compiler that generates code for a statement, or group of statements, which is independent of the code generated for other...

15+ min read

Syntax Directed Translation in Compiler Design

Syntax-Directed Translation (SDT) is a method used in compiler design to convert source code into another form while analyzing its structure. It...

15+ min read

Semantic Analysis is the third phase of Compiler. Semantic Analysis makes sure that declarations and statements of program are semantically correct. I...

10 min read

Storage Allocation Strategies in Compiler Design

A compiler is a program that converts HLL(High-Level Language) to LLL(Low-Level Language) like machine language. It is also responsible for...

15+ min read

Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:

K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

Placement Training Program

GeeksforGeeks Videos
Geeks Community

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
DSA Interview Questions
Competitive Programming

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
NodeJs
Bootstrap
Tailwind CSS

Python Tutorial

Python Programming Examples
Django Tutorial
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question

Computer Science

GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

DevOps

Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Chemistry
Biology
Social Science
English Grammar

Accounting Software Directory
HR Management Tools
Editing Software Directory
Microsoft Products and Apps
Figma Tutorial

Databases

SQL
MYSQL
PostgreSQL
PL/SQL
MongoDB

Preparation Corner

Company-Wise Recruitment Process
Resume Templates
Aptitude Preparation
Puzzles
Company-Wise Preparation
Companies
Colleges

Competitive Exams

JEE Advanced
UGC NET
UPSC
SSC CGL
SBI PO
SBI Clerk
IBPS PO
IBPS Clerk

More Tutorials

Software Development
Software Testing
Product Management
Project Management
Linux
Excel
All Cheat Sheets
Recent Articles

Free Online Tools

Typing Test
Image Editor
Code Formatters
Code Converters
Currency Converter
Random Number Generator
Random Password Generator

Write & Earn

Write an Article
Improve an Article
Pick Topics to Write
Share your Experiences
Internships

DSA/Placements

DSA - Self Paced Course
DSA in JavaScript - Self Paced Course
DSA in Python - Self Paced
C Programming Course Online - Learn C with Data Structures
Complete Interview Preparation
Master Competitive Programming
Core CS Subject for Interview Preparation
Mastering System Design: LLD to HLD
Tech Interview 101 - From DSA to System Design [LIVE]
DSA to Development [HYBRID]
Placement Preparation Crash Course [LIVE]

Development/Testing

JavaScript Full Course
React JS Course
React Native Course
Django Web Development Course
Complete Bootstrap Course
Full Stack Development - [LIVE]
JAVA Backend Development - [LIVE]
Complete Software Testing Course [LIVE]
Android Mastery with Kotlin [LIVE]

Machine Learning/Data Science

Complete Machine Learning & Data Science Program - [LIVE]
Data Analytics Training using Excel, SQL, Python & PowerBI - [LIVE]
Data Science Training Program - [LIVE]
Mastering Generative AI and ChatGPT
Data Science Course with IBM Certification

Programming Languages

C Programming with Data Structures
C++ Programming Course
Java Programming Course
Python Full Course

Clouds/Devops

DevOps Engineering
AWS Solutions Architect Certification
Salesforce Certified Administrator Course

GATE 2026

GATE CS Rank Booster
GATE DA Rank Booster
GATE CS & IT Course - 2026
GATE DA Course 2026
GATE Rank Predictor
