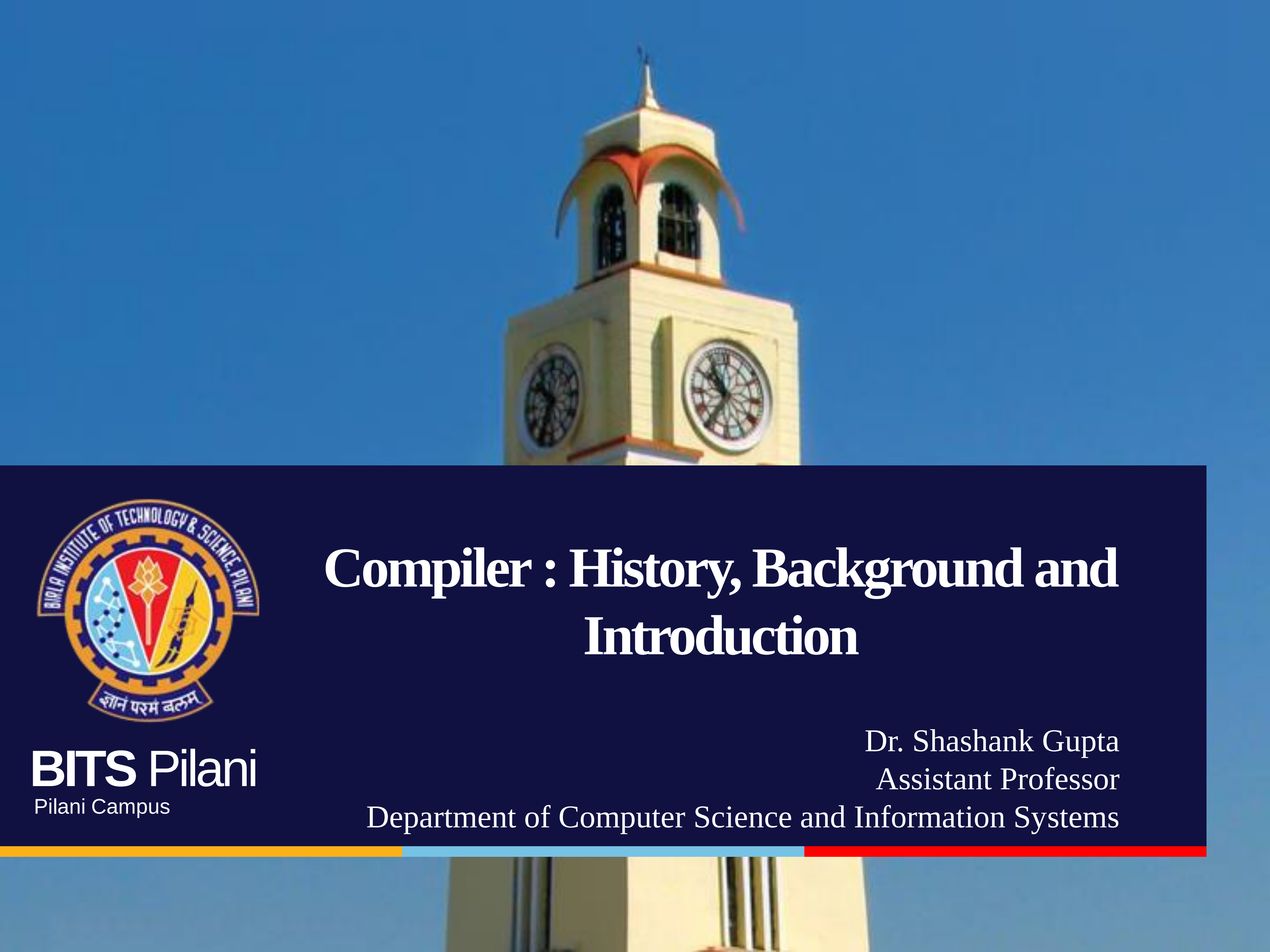


BITS Pilani
Pilani Campus

# Compiler : History, Background and Introduction

Dr. Shashank Gupta
Assistant Professor
Department of Computer Science and Information Systems

## Background Prerequisites

A reasonable amount of basic computer science knowledge is desirable.

Basic working knowledge of C Programming, Data structures and Theory of Computation, etc.

## References

Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. "Compilers, principles, techniques." *Addison wesley* 7, no. 8 (1986): 9.

- **a.k.a. Dragon Book**

Andrew W. Appel, “Modern Compiler Implementation in C”, Cambridge University Press

Ravi Sethi, “Programming Languages - Concepts and Constructs”, Pearson Education

The course content (i.e. lecture slides, pointers to reading material, etc.) will be available to you via the course website.

## Logistics of this Course

Please keep in touch with the course website https://nalanda.bits-pilani.ac.in/login/index.php by selecting the CS F363.

Give feedback through out the semester. Feel free to discuss anything with me.

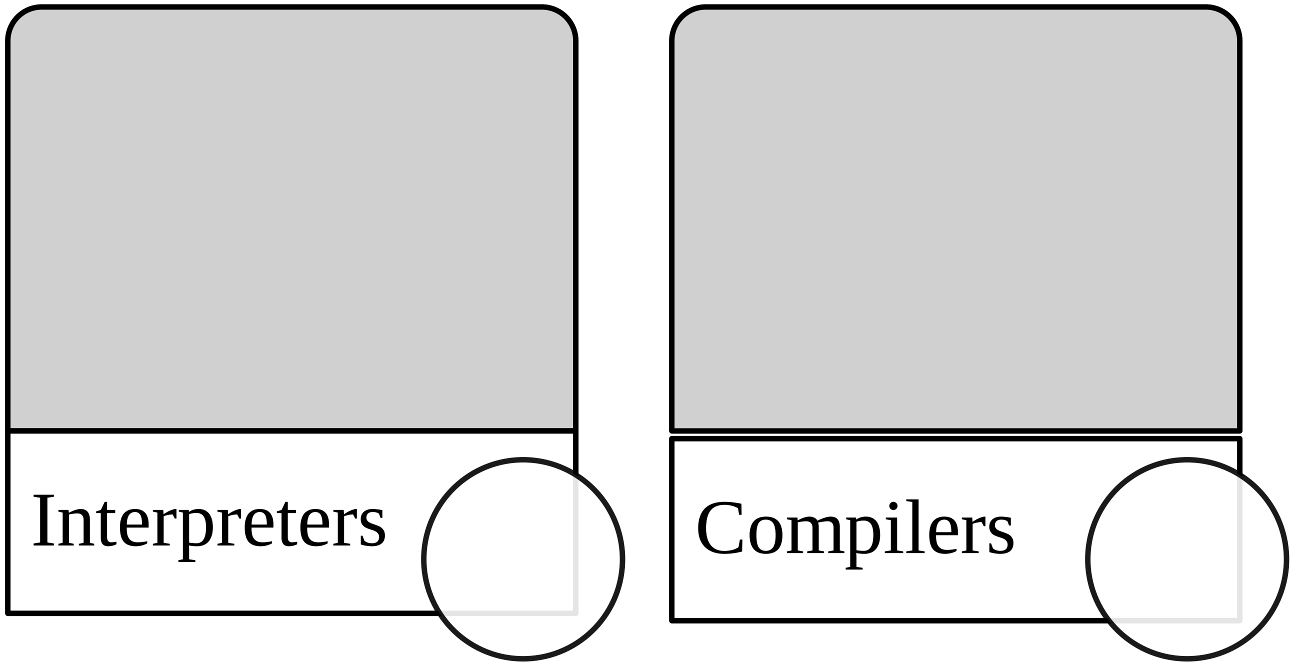
## Today's Agenda

### Key Points to be Covered

- History of Compiler
- What are Compilers?
- What does Compiler do ?

## History

In late 1950s, two key strategies were utilized for implementing the programming languages.



## History (Continued…..)

Some platforms include both interpreter and compiler

For e.g. Lisp, Java

## Late Inventions of 1950s

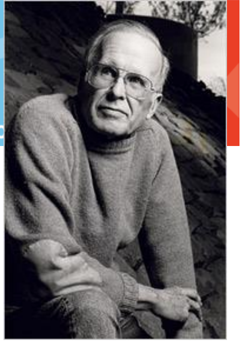
IBM developed 704 in 1954.

- Cost of s/w development is much higher than cost of h/w.

Speedcoding Interpreter

- Program execution is very slower.

## First Compiler: Fortran I Project (1954-57)



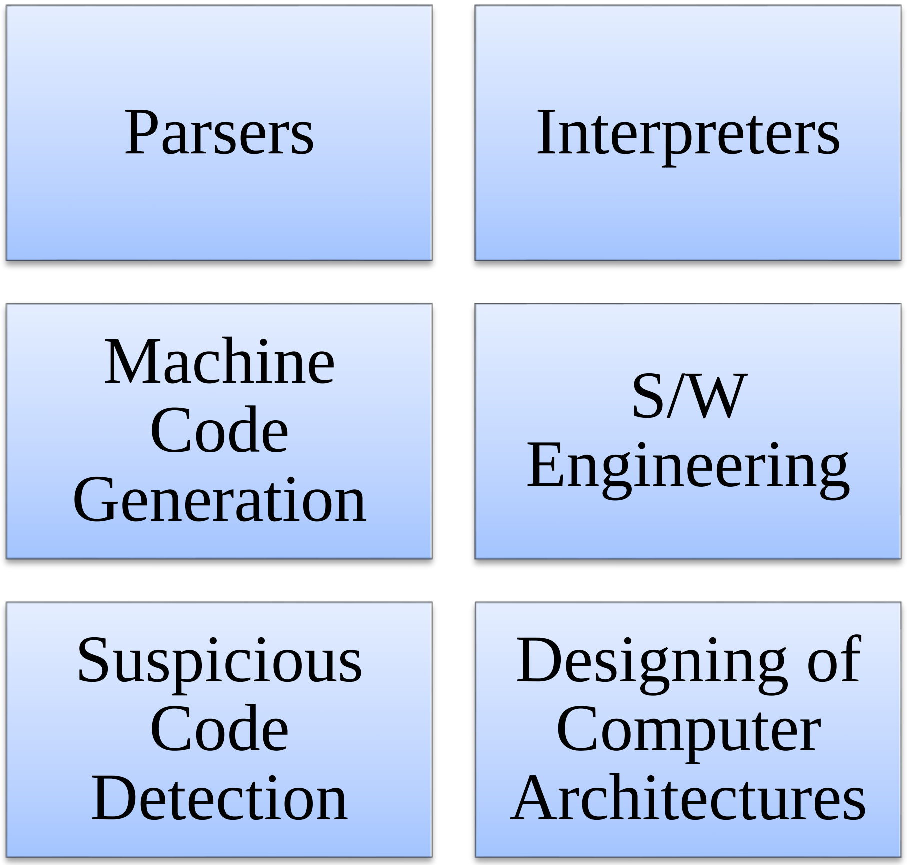
It was developed by John Backus (Turing awardee)

More than half of the programmers were started using Fortran by 1958.

Present compilers follow the basic structure of this compiler.

# Motivation for Studying the Compiler

Numerous Applications for Compiler Technology.


Parsers
Interpreters
Machine Code Generation
S/W Engineering
Suspicious Code Detection
Designing of Computer Architectures

## What Does Compilers do?

## What Does Compilers do?

Converts one form of program to another form.

Usually from high level source code language to low level machine code.

## What Does Compilers do?

**Source code** is optimized for human readability.

- It should be expressive and redundant to omit errors.

Whereas, **machine code** is optimized for resultant hardware.

- Redundancy is reduced and intent about the code is somewhat lost.

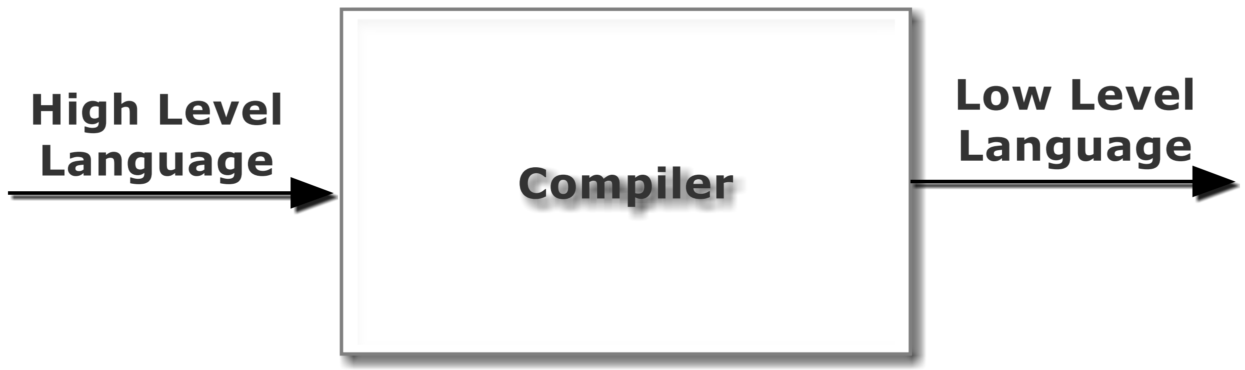
## Process of Translation

Source code and machine code mismatch in terms of level of abstraction.

- Source code deals with the variables, operators, expressions, etc. whereas machine code deals with memory, registers, stack, opcodes, addressing modes, etc.

Some source languages are too closer from machine languages, while some are not.

## Compiler



# Compiler

**Abstraction at Source Code Level**

- Identifiers
- Operators
- Expressions
- Statements
- Conditionals
- Iterations
- Functions

**Abstractions at Target Code Level**

- Memory Locations
- Registers
- Stack
- Opcodes
- Addressing Modes
- System Libraries

## Motto of Translation

- Efficient Performance related to Compile Time.
- Correctness of Compiler
- Maintainable Code

## Process of Translation

## Process of Translation

Translate in appropriate steps.

Generate a chain of valid intermediate representations.

Such representations must be easily modifiable to any kind of changes in program.

# Steps in Lexical Analysis

1. - Identify the valid set of Characters in language.

2. - Break the sequence of characters in appropriate words **(Tokens).**
   - Keywords, Identifiers, Numbers, Operators, Punctuation Symbols, etc.

3. - Find out whether these tokens are valid or not.

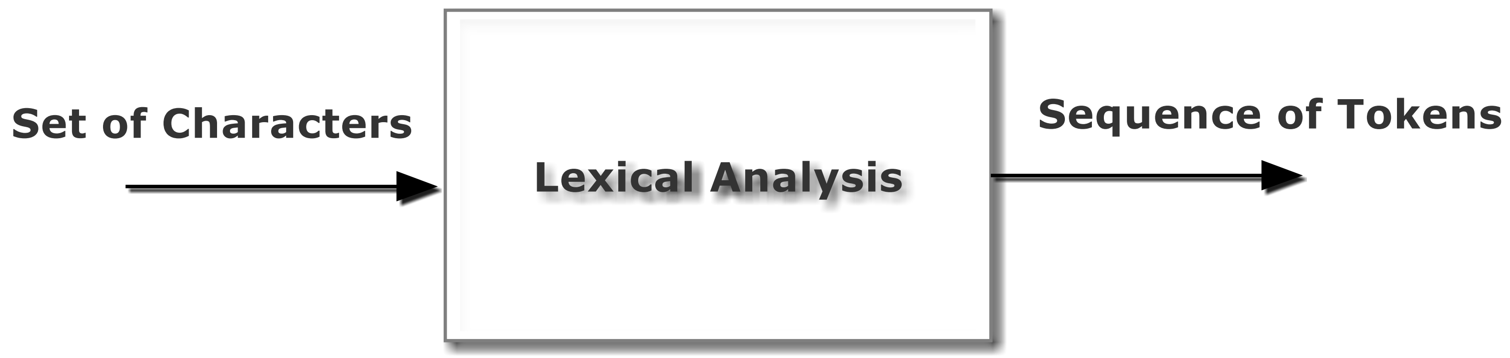
## Lexical Analysis

Identifying/Spotting tokens is not as simple.

- For e.g. thi si st hefir s tmo dul eofcomp iler

Hence, compiler should know the process of recognizing word boundaries.

# Lexical Analysis



## Lexical Analysis

Hence, the key goal of lexical analyzer is to break a sentence into a series of words/tokens.

- White Spaces and Punctuation Symbols are usually considered to be word separators in English Language.

Most of the times, even a single character from a different class can also be identified as a word boundary

- Input: ifp == qthenflag=1;
- Output: Sequence of Tokens (9 Tokens): if p == q then flag = 1;

## Lexical Analysis

Identify the token and ignore the white spaces, comments, etc.

Reports the error.

Rules will be designed using regular expressions that would be recognized using Finite State Machine.

## Lexical Phase Errors

Occurrence of illegal characters.

Exceeding length of identifier.