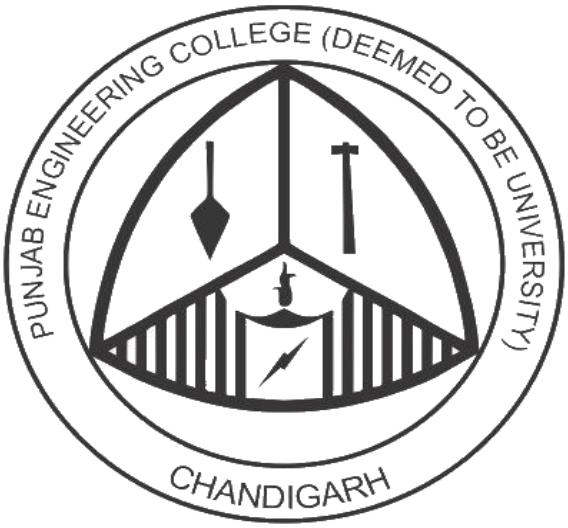
# PUNJAB ENGINEERING COLLEGE

(Deemed To Be University)




## SOFTWARE ENGINEERING

CSN 302

---

# PROJECT SUMMARY

---

**Submitted To:**
Mr. Rajesh Bhatia

**Submitted By:**
Srishti Arora(18103011)
Jasmeen Bansal(18103016)
Bhavya Gulati(18103024)
Saksham Basandrai(18103048)

# DeCipher

"I hear and I forget. I see and I remember. I do and I understand."

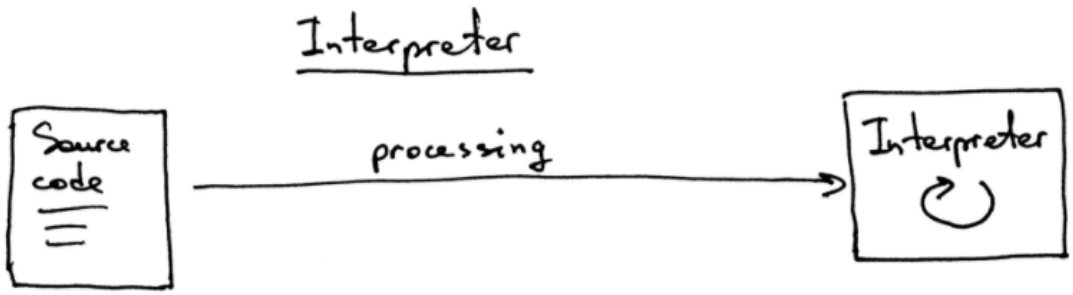
*-Confucius*

These famous lines said by Confucius, holds true in all scenarios. For the best way to fully test the understanding of a concept is by building or creating it and getting our own hands dirty in the process.

So, following the same philosophy we attempted to build our very own **Interpreter** from scratch, so as to gain a better insight on how it works and functions.

We generally write a computer program using a high-level language. A high-level language is one that is understandable by us, humans. This is called **source code**. However, a computer does not understand high-level language. It only understands the program written in 0's and 1's in binary, called the **machine code.**

To convert source code into machine code, we use either a compiler or an interpreter. To avoid the hassle of converting to machine code, an interpreter directly executes the source program without translating it into machine language first.



We are going to create a simple interpreter for a large subset of **Pascal language** and a source-level debugger which will be **implemented in C++.**

When we feed a source program into an interpreter, it takes over to check and execute the program. Since the interpreter is in control when it is executing the source program, when it encounters an error it can stop and display a message containing the line number of the offending statement and the name of the variable. It can even prompt the user for some corrective action before resuming execution of the program.

During the course of this project, we wish to implement the following **features:**

- Evaluating complex arithmetic expressions
- Checking associativity and precedence
- Assignment and Unary operator's evaluation
- Evaluating compound statements
- Semantic Analysis
- Procedure and function calls
- Solving nested procedures and functions

Interpreters are complex programs, and writing them successfully is hard work. To tackle the complexity, a strong software engineering approach can be used. **Design patterns, Unified Modeling Languages (UML) diagrams**, and other modern object-oriented design practices make the code understandable and manageable.