Mini-Assignment #3
CS3423: Compilers-2
Transpilers for Indic Languages

Names: Vikhyath Sai Kothamasu, Taha Adeel Mohammed, Pranav K Nayak
Roll Numbers: cs20btech11056, cs20btech11052, es20btech11035

Q1) Problem Statement: Based on the video, write a very brief summary of the problem statement that is being solved.

A1) With the advent of Unicode inputting Indic text onto the computer and smartphones, it has become ever easier to type in Indic languages. It can get tricky for people new to programming to learn a complicated language with a new syntax that is not quite human-readable at the moment. There are languages such as python which come close to human readable but there is still a significant amount of work left in this domain. And especially for people who are not familiar with English, it can get more difficult. So for such people, it can be one hurdle less if they do not have to learn English and yet get started with coding in their own language. It also makes more sense to transpile it to an existing language such as TypeScript because eventually, the computers understand only binary language, and the process that remains after the lexical analysis and parsing is the same instead of writing separate large codebases for the same purpose. A step forward in the project would be to let the user code identifier names and literals such as numbers also in their native language. It might be a challenge to translate string literals as they require Natural language processing concepts and context as some languages have words that have multiple meanings based on their use in the given context. Given that there are already multiple open-sourced transpilers in JavaScript and other languages, it helps that this transpiler is TypeScript based as it is the superset of Javascript and has several features such as static typing, OOPS, and many more as mentioned in the presentation.

Q2) Examples: Write 5 correct and 5 incorrect programs using the above interface in your mother tongue.
Advantages/Disadvantages: List out the advantages and disadvantages of using transpilers.

A2) Examples are provided in the attached tar file.

Some advantages of using a transpiler are:

- A transpiler may translate its language to the target language, which is then compiled using the compiler for the target language to produce native or virtual machine code. Consequently, the benefits of the target compiler are immediately passed on to the transpiler. It basically acts like inheritance but for an entire high level language.
- It is straightforward to translate this structured intermediate code into any other language using a simple translator since the transpiler creates structured, human-readable intermediate code.
- Transpiler is a fantastic tool for learning programming since it offers condensed syntaxes and structures.
- The target language's more complex constructions can be reduced and written in the translator's language as more compact structures. This would lessen the quantity of the amount of typing required to complete the task and decrease its complexity.
- Since it doesn't need to be done manually, it cuts down on the time it takes to translate source code across languages.
- Produce code that complies with an earlier version while developers take advantage of features from a later version, as is the case with browsers that aren't up to date with the most recent JavaScript standards.

Some disadvantages of using a transpiler are:

- None of the programming languages created by humans are exactly the same. There is at least one important difference between them and as a result, it can never be a perfect transpilation of code from one language to another. Sometimes, the code involving this unique feature might have to be omitted during transpilation.
- When one is transpiling code from an interpreted language (dynamically typed) to compiled language (statically typed), they might face issues because, in an interpreted language, a variable with the same name can be redeclared with another data type whereas the same is not possible in the compiled language. The transpiler might have to use a new identifier for the same and remember to use it from there on.

Q3) Motivation: Describe realistic scenarios when the above infrastructure could be useful.

a) **Propose different ways assisting the programmers to improve the coding productivity.**

A) To improve the coding productivity of programmers while coding in Indic languages in IDEs, the following features can be implemented for the IDE:

- As keyboard support for Indic languages is lacking compared to the Latin script, the IDEs can support phonetic translation from English to the Indic keywords (Eg: maan to मान).
- Support for IntelliSense and autocomplete suggestions can be added to greatly increase the productivity of the programmers.
- The error debugging messages generated by the transpiler can also be in the indic language, allowing easier debugging for the programmers.
- Also while the current transpiler just supports Indic keywords, adding support for indic variable names and numbers would increase productivity as the programmer wouldn't have to constantly switch between languages.

b) **Propose a set of sample apps that can be built using Indic language.**

A) Indic languages would greatly increase the reach of programming languages, allowing anyone to start coding, and removing the prerequisite of having a proper grasp of English before they can code. Hence these languages can be used by the common man to comfortably build applications for their daily tasks. A few such examples are:

- Shopkeepers can easily build apps for their shop, allowing customers to easily browse their products and place orders, and also maybe offer home delivery services
- Apps can be built for barber shops/hair stylers that would access the user's camera and use filters to show what a haircut would look like.
- Apps for furniture shops that would allow users to browse the furniture and view their exact dimensions easily, hence letting them know the space requirements
- Local grocers/butchers/etc can build apps that would mostly automate the process of interacting with their suppliers, hence making the process easier for both the shopkeepers and the producers.

Q4) Transpiler Implementation:

a) **Can we extend this transpiler to generate the code native? If yes, how?**

A) Yes, this transpiler, or any transpiler in general, can be easily extended to generate the code natively.

The transpiler is implemented such that it reads the Indic language and generates the token stream according to the language keywords. The token stream is parsed using the grammar rules of the language to generate the AST (Abstract Syntax Tree) for the code. This AST is converted into TypeScript code, which is our final output.

To generate the code native, instead of converting the AST to TypeScript code, we can extend the transpiler to convert the AST to an Intermediate Representation (possibly LLVM IR) and then run the code generation phase on the IR to generate the code natively.
Another alternative to generate the code native would be to use the compiler for TypeScript to compile the output of our transpiler into JavaScript while asserting that the variable types are right. The compiled JavaScript code can be interpreted to generate the code natively.

b) **Is there any implication on performance on using the transpiler? If yes, then list down the ways to improve the same or mitigate the same.**

A) When using a transpiler for our language, we run the lexical analyzer, the syntax analyzer, and the semantic analyzer to get the parse tree, from which we generate the TypeScript code. This TypeScript code would again undergo the lexical, syntax, and semantic analysis phases before undergoing the remaining compiling phases. Hence there is a performance overhead as the same phases are being run multiple times and there is redundancy in the phases.

We can improve and mitigate these performance costs by directly generating the code native from the parse tree instead of converting it into TypeScript code and then using TypeScript's compiler.

c) **From the way it is implemented do you see if we can improve the code better in terms of making it less buggy and absorbing the changes faster?**

A) The current transpiler maps the Sanskrit keywords to their corresponding keyword in TypeScript and the lexer, parser, semantic analyzer, and other code reflect this mapping. While this works for one language, to make the project more easily extensible and allow easier and lesser buggy change absorption, an interface can be provided to write the mapping from our native Indic language to TypeScript, and the code for the lexer, parser, etc is automatically generated to support our new mapping. This would also reduce the chances of human errors while coding, as the code is generated automatically. This automation is also quite practical, as the basic structure and design of the lexer, parser, etc is mostly the same for the different Indic languages.