Bangla Script: A Bengali Programming Language

Description: BanglaScript is a programming language that uses Bengali characters for its syntax. It was a fun project and I started to learn more about how programming languages work. The goal of this project is to create a simple programming language that can be used to write simple programs. The language is still in its early stages, and there is a lot of work to be done. However, I am excited to see where this project goes.

Tech Stack:

1. React with Vite
2. Typescript
3. ohmJS

Folder Structure:

```
dist/
public/
src/
 |_ assets/
 |_ pages/
 | |_Editor.tsx          // Code editor where the user will write a Bengali syntax program.
 | |_Home.tsx            // Project Details
 | |_Docs.tsx            //Documentation, How user can write the program in Bengali.
 |_ App.css
 |_ App.tsx
 |_ index.css
 |_ main.tsx
 |_ grammar/
 | |_statements.ts       // Grammar and semantics Print and statements
 | |_variables.ts        // Grammar and semantics for variable, types
 | |_conditionals.ts     // Grammar and semantics for conditional statements. Will use
 | |_ loops.ts           // Grammar and semantics for loops
 | |_ oop.ts             // Grammar and semantics for oop syntax
 |_ parser.ts            // Centralized semantics file with parsing functionality
 |_ index.tsx            // Main entry point
vite-env.d.ts
.gitignore
README.md
eslint.config.js
index.html
package-lock.json
```

package.json
tsconfig.app.json
tsconfig.json
tsconfig.node.json
vite.config.ts

Keep in mind that we will not write the print and statements, variables grammar again in the conditional block. We will use the existing grammar and semantics on the conditional.ts file. As long as in a block of loop statements we have a dependency of print and statements, variables, conditional grammar, and semantics. So we will use them in loop.ts. Last but not least oop or class declaration concept has all the dependencies mentioned earlier. We will use all grammar and semantics in the oop.ts with its opp grammar and semantics. Centralized semantics(parser.ts) will be used in Editor.tsx as a transpiler of the project.

Some important codes:

parser.ts

```
import * as ohm from 'ohm-js';
import fs from 'fs';
import path from 'path';

// Load grammar from files dynamically
const grammarPath = path.join(__dirname, 'grammars', 'main.ohm');
const banglaGrammar = fs.readFileSync(grammarPath, 'utf-8');

// Create the Ohm grammar object
const BanglaScript = ohm.grammar(banglaGrammar);
const semantics = BanglaScript.createSemantics();

/**
 * Define the semantics for transpiling BanglaScript to TypeScript
 */
semantics.addOperation('toTS()', {
  Program(statements) {
    return statements.children.map((s) => s.toTS()).join("\n");
  },
  ConsoleLogStatement(_write, _openParen, str, _closeParen, _semicolon) {
    return `console.log(${str.toTS()});`;
  },
```

```
  VariableDeclaration(type, name, _eq, value, _semicolon) {
    const tsType = {
      "সংখ্যা": "number",
      "হাছামিছা": "boolean",
      "দড়ি": "string",
      "বিন্যাস": "array"
    }[type.sourceString] || "any";
    console.log(`ধরি ${name.sourceString}: ${tsType} = ${value.toTS()};`);
    return `ধরি ${name.sourceString}: ${tsType} = ${value.toTS()};`;
  },
  FunctionDeclaration(_func, name, _open, params, _close, _openBody, body, _closeBody) {
    return `function ${name.sourceString}(${params.toTS()}) {\n${body.toTS()}\n}`;
  },
  IfElseStatement(_if, _open, condition, _close, _openBody, body, _closeBody) {
    return `if (${condition.toTS()}) {\n${body.toTS()}\n}`;
  },
  LoopStatement_forLoop(_for, _open, init, _semi1, condition, _semi2, increment, _close,
_openBody, body, _closeBody) {
    return `for (${init.toTS()}; ${condition.toTS()}; ${increment.toTS()}) {\n${body.toTS()}\n}`;
  },
  LoopStatement_whileLoop(_while, _open, condition, _close, _openBody, body, _closeBody) {
    return `while (${condition.toTS()}) {\n${body.toTS()}\n}`;
  },
  LoopStatement_doWhileLoop(_do, _openBody, body, _closeBody, _while, _open, condition,
_close, _semi) {
    return `do {\n${body.toTS()}\n} while (${condition.toTS()});`;
  },
  ClassDeclaration(_class, name, _openBody, body, _closeBody) {
    return `class ${name.sourceString} {\n${body.toTS()}\n}`;
  },
  string(_open, chars, _close) {
    return `"${chars.sourceString}"`;
  },
  identifier(chars) {
    return chars.sourceString;
  },
  number(chars) {
    return chars.sourceString;
  },
  boolean(chars) {
```

```
    if(chars.sourceString === "সত্য") {
      return "true";
    }
    else if(chars.sourceString === "মিথ্যা") {
      return "false";
    }
    else {
      throw new Error("আপনি ভুল বুলিয়ান মান দিয়েছেন: " + chars.sourceString);
    }
  }
});

/**
 * Function to transpile BanglaScript code to TypeScript
 * @param code - BanglaScript code
 */
export default function transpileBanglaScript(code: string): string {
  const matchResult = BanglaScript.match(code);
  if (matchResult.failed()) {
    throw new Error("Syntax Error: " + matchResult.message);
  }
  return semantics(matchResult).toTS();
}


transliterate.ts
/**
 * Transliterates Bangla variable names to Latin script (English)
 * to ensure valid TypeScript variable names.
 * @param text - Bangla text to transliterate
 * @returns Transliterated string
 */
export function transliterateBangla(text: string): string {
  const transliterationMap: Record<string, string> = {
          "অ": "a", "আ": "aa", "ই": "i", "ঈ": "ii", "উ": "u", "ঊ": "uu",
          "ঋ": "ri", "এ": "e", "ঐ": "oi", "ও": "o", "ঔ": "ou",
          "ক": "k", "খ": "kh", "গ": "g", "ঘ": "gh", "ঙ": "ng",
          "চ": "ch", "ছ": "chh", "জ": "j", "ঝ": "jh", "ঞ": "ny",
          "ট": "t", "ঠ": "th", "ড": "d", "ঢ": "dh", "ণ": "n",
          "ত": "t", "থ": "th", "দ": "d", "ধ": "dh", "ন": "n",
```

```
        "প": "p", "ফ": "ph", "ব": "b", "ভ": "bh", "ম": "m",
        "য": "j", "র": "r", "ল": "l", "শ": "sh", "ষ": "s",
        "স": "s", "হ": "h", "ড়": "r", "ঢ়": "rh", "য়": "y",
        "ৎ": "t", "ং": "ng", "ঃ": "h"
  };

  return text
        .split("")
        .map(char => transliterationMap[char] || char)
        .join("")
        .replace(/[^a-zA-Z0-9_]/g, ""); // Remove invalid characters
}

statement.ts
import * as ohm from 'ohm-js';

const statementsGrammar = `
  Statements {
        Statement = PrintStatement | VariableDeclaration

        PrintStatement = "দেখাও" "(" string ")" ";"
        VariableDeclaration = "ধরি" Identifier "=" Expression ";"

        Expression = "সত্য" | "মিথ্যা" | string | number
        string = "\\"" (~"\\"" any)* "\\""
        number = digit+
        Identifier = letter (letter | digit)*
  }
`;

export const Statements = ohm.grammar(statementsGrammar);
export const statementsSemantics = Statements.createSemantics();

statementsSemantics.addOperation('toTS()', {
  Statement(stmt) { return stmt.toTS(); },
  PrintStatement(_write, _open, str, _close, _semi) {
        return `console.log(${str.toTS()});`;
  },
  VariableDeclaration(_let, name, _eq, value, _semi) {
        return `let ${name.sourceString} = ${value.toTS()};`;
```

```
  },
  string(_open, chars, _close) {
        return `"${chars.sourceString}"`;
  },
  number(n) {
        return n.sourceString;
  }
});

import * as ohm from 'ohm-js';

const statementsGrammar = `
  Statements {
        Statement = PrintStatement | VariableDeclaration

        PrintStatement = "দেখাও" "(" string ")" ";"
        VariableDeclaration = "ধরি" Identifier "=" Expression ";"

        Expression = "সত্য" | "মিথ্যা" | string | number
        string = "\\"" (~"\\"" any)* "\\""
        number = digit+
        Identifier = letter (letter | digit)*
  }
`;

export const Statements = ohm.grammar(statementsGrammar);
export const statementsSemantics = Statements.createSemantics();

statementsSemantics.addOperation('toTS()', {
  Statement(stmt) { return stmt.toTS(); },
  PrintStatement(_write, _open, str, _close, _semi) {
        return `console.log(${str.toTS()});`;
  },
  VariableDeclaration(_let, name, _eq, value, _semi) {
        return `let ${name.sourceString} = ${value.toTS()};`;
  },
  string(_open, chars, _close) {
        return `"${chars.sourceString}"`;
  },
  number(n) {
```

```
        return n.sourceString;
  }
});

variables.ts
import * as ohm from 'ohm-js';
import { transliterateBangla } from "./transliterate.ts";

/**
 * Define Ohm grammar for variable declarations in BanglaScript
 */
const variableGrammar = `
Variables {
  Program = Statement*
  Statement = VariableDeclaration

  VariableDeclaration = "ধরি" VarType identifier "=" Expression ";"
  VarType = "সংখ্যা" | "হাছামিছা" | "দড়ি" | "বিন্যাস" | "সংখ্যা_বিন্যাস" | "দড়ি_বিন্যাস"
  Expression = number | boolean | string | identifier | ArrayExpression

  ArrayExpression = "[" ListOf<Expression, ","> "]"

  identifier = letter (letter | digit | "_")*
  number = digit+
  boolean = "সত্য" | "মিথ্যা"
  string = "\\"" (~"\\"" any)* "\\""
}
`;

/**
 * Create a new Ohm grammar
 */
const Variables = ohm.grammar(variableGrammar);
const semantics = Variables.createSemantics();

/**
 * Define semantics for variable declarations
 */
semantics.addOperation('toTS()', {
  Program(statements) {
```

```
        return statements.children.map((s) => s.toTS()).join("\n");
  },

  VariableDeclaration(_dhori, type, name, _eq, value, _semicolon) {
        const tsTypeMap: Record<string, string> = {
        "সংখ্যা": "number",
        "হাছামিছা": "boolean",
        "দড়ি": "string",
        "বিন্যাস": "any[]",
        "সংখ্যা_বিন্যাস": "number[]",
        "দড়ি_বিন্যাস": "string[]"
        };

        // Validate type
        const typeString = type.sourceString;
        if (!(typeString in tsTypeMap)) {
        throw new Error(`ত্রুটি: '${typeString}' কোন বৈধ ডাটা টাইপ নয়!`);
        }

        // Transliterate Bengali variable name
        const varName = transliterateBangla(name.sourceString);

        // Validate variable name (must start with a letter or _ and contain only alphanumeric
characters or _)
        if (!varName.match(/^[a-zA-Z_][a-zA-Z0-9_]*$/)) {
        throw new Error(`ত্রুটি: '${name.sourceString}' একটি অবৈধ ভেরিয়েবল নাম! ইংরেজি বর্ণমালা
বা "_" ব্যবহার করুন।`);
        }

        return `let ${varName}: ${tsTypeMap[typeString]} = ${value.toTS()};`;
  },

  ArrayExpression(_open, elements, _close) {
        return `[${elements.children.map((e) => e.toTS()).join(", ")}]`;
  },

  identifier(name) {
        return transliterateBangla(name.sourceString);
  },
```

```
  number(value) {
        return value.sourceString;
  },

  boolean(value) {
        return value.sourceString === "সত্য" ? "true" : "false";
  },

  string(_open, chars, _close) {
        return `"${chars.sourceString}"`;
  }
});

/**
 * Function to transpile BanglaScript variable declaration to TypeScript
 */
export default function transpileVariables(code: string): string {
  const matchResult = Variables.match(code);
  if (matchResult.failed()) {
        throw new Error("ত্রুটি: " + matchResult.message);
  }
  return semantics(matchResult).toTS();
}

conditionals.ts
import * as ohm from 'ohm-js';
import { transliterateBangla } from "./transliterate.ts";

/**
 * Define Ohm grammar for conditional statements in BanglaScript
 */
const conditionalsGrammar = `
Conditionals {
  Program = Statement*
  Statement = IfStatement | BlockStatement
  IfStatement = "যদি" "(" Expression ")" BlockStatement ElseIfStatement* ElseStatement?
  ElseIfStatement = "নয়তোযদি" "(" Expression ")" BlockStatement
  ElseStatement = "নয়তো" BlockStatement
  BlockStatement = "{" Statement* "}"
```

```
  Expression = boolean | identifier | Comparison
  Comparison = Expression ComparisonOp Expression
  ComparisonOp = "==" | "!=" | ">" | "<" | ">=" | "<="

  identifier = letter (letter | digit | "_")*
  boolean = "সত্য" | "মিথ্যা"
}
`;

const Conditionals = ohm.grammar(conditionalsGrammar);
const semantics = Conditionals.createSemantics();

semantics.addOperation('toTS()', {
  Program(statements) {
        return statements.children.map((s) => s.toTS()).join("\n");
  },

  IfStatement(_if, _open, condition, _close, ifBlock, elseIfs, elseStmt) {
        return `if (${condition.toTS()}) ${ifBlock.toTS()}\n` +
        elseIfs.children.map((e) => e.toTS()).join("\n") +
        (elseStmt.numChildren ? `\n${elseStmt.toTS()}` : "");
  },

  ElseIfStatement(_elseif, _open, condition, _close, block) {
        return `else if (${condition.toTS()}) ${block.toTS()}`;
  },

  ElseStatement(_else, block) {
        return `else ${block.toTS()}`;
  },

  BlockStatement(_open, statements, _close) {
        return `{\n  ${statements.children.map((s) => s.toTS()).join("\n  ")}\n}`;
  },

  Comparison(left, op, right) {
        return `${left.toTS()} ${op.sourceString} ${right.toTS()}`;
  },

  identifier(name) {
```

```
        return transliterateBangla(name.sourceString);
  },

  boolean(value) {
        return value.sourceString === "সত্য" ? "true" : "false";
  }
});


export default function transpileConditionals(code: string): string {
  const matchResult = Conditionals.match(code);
  if (matchResult.failed()) {
        throw new Error("ত্রুটি: " + matchResult.message);
  }
  return semantics(matchResult).toTS();
}

loops.ts
import * as ohm from "ohm-js";
import { transliterateBangla } from "./transliterate";

const loopsGrammar = `
  Loops {
        LoopStatement = ForLoop | WhileLoop | DoWhileLoop

        ForLoop = "জন্য" "(" VariableDeclaration Condition ";" Assignment ")" Block
        WhileLoop = "যতক্ষণ" "(" Condition ")" Block
        DoWhileLoop = "কর" Block "যতক্ষণ" "(" Condition ")" ";"

        Condition = identifier Operator identifier
        Operator = "==" | "!=" | "<" | ">" | "<=" | ">="

        Assignment = identifier "=" identifier Operator? identifier?

        Block = "{" Statement* "}"
        Statement = VariableDeclaration | Assignment | LoopStatement | PrintStatement

        VariableDeclaration = VarType identifier "=" Expression ";"
        VarType = "সংখ্যা" | "হাছামিছা" | "দড়ি" | "বিন্যাস"
        Expression = number | boolean | string | identifier
```

```
        identifier = letter (letter | digit)*
        number = digit+
        boolean = "সত্য" | "মিথ্যা"
        PrintStatement = "দেখাও" "(" string ")" ";"
        string = "\\"" (~"\\"" any)* "\\""
  }
`;

const Loops = ohm.grammar(loopsGrammar);
const semantics = Loops.createSemantics();

/**
 * Semantic rules for loops
 */
semantics.addOperation("toTS()", {
  ForLoop(_for, _open, init, condition, _semi, assignment, _close, block) {
        return `for (${init.toTS()} ${condition.toTS()}; ${assignment.toTS()}) ${block.toTS()}`;
  },

  WhileLoop(_while, _open, condition, _close, block) {
        return `while (${condition.toTS()}) ${block.toTS()}`;
  },

  DoWhileLoop(_do, block, _while, _open, condition, _close, _semi) {
        return `do ${block.toTS()} while (${condition.toTS()});`;
  },

  Condition(left, operator, right) {
        return `${transliterateBangla(left.sourceString)} ${operator.sourceString}
${transliterateBangla(right.sourceString)}`;
  },

  Assignment(name, _eq, value, op, extra) {
        let assignment = `${transliterateBangla(name.sourceString)} =
${transliterateBangla(value.sourceString)}`;
        if (op.sourceString) {
        assignment += ` ${op.sourceString} ${transliterateBangla(extra.sourceString)}`;
        }
        return assignment + ";";
```

```
  },

  Block(_open, statements, _close) {
        return `{\n${statements.children.map(s => s.toTS()).join("\n")}\n}`;
  },

  VariableDeclaration(type, name, _eq, value, _semicolon) {
        const tsType = {
        "সংখ্যা": "number",
        "হাছামিছা": "boolean",
        "দড়ি": "string",
        "বিন্যাস": "any[]"
        }[type.sourceString] || "any";

        return `let ${transliterateBangla(name.sourceString)}: ${tsType} = ${value.toTS()};`;
  },

  PrintStatement(_print, _open, str, _close, _semicolon) {
        return `console.log(${str.toTS()});`;
  },

  string(_open, chars, _close) {
        return `"${chars.sourceString}"`;
  }
});

/**
 * Function to transpile BanglaScript loops to TypeScript
 * @param code - BanglaScript code
 */
export default function transpileLoops(code: string): string {
  const matchResult = Loops.match(code);
  if (matchResult.failed()) {
        throw new Error("Syntax Error in loop statement: " + matchResult.message);
  }
  return semantics(matchResult).toTS();
}
```

oop.ts

```
import * as ohm from "ohm-js";
```

```
import { transliterateBangla } from "./transliterate";
import transpileVariables from "./variables";  // For variable declaration in OOP
import transpileStatements from "./statements"; // Importing statements
import transpileConditionals from "./conditionals"; // Importing conditionals
import transpileLoops from "./loops"; // Importing loops

const oopGrammar = `
  OOP {
        OOPStatement = ClassDeclaration | ObjectInstantiation | MethodCall

        ClassDeclaration = "শ্রেণী" identifier "{" VariableDeclaration* ConstructorDeclaration?
MethodDeclaration* "}"
        ConstructorDeclaration = "নির্মাতা" "(" ParameterList? ")" Block
        MethodDeclaration = "পদ্ধতি" identifier "(" ParameterList? ")" Block
        ObjectInstantiation = identifier "=" "নতুন" identifier "(" ArgumentList? ")" ";"
        MethodCall = identifier "." identifier "(" ArgumentList? ")" ";"

        ParameterList = identifier ("," identifier)*
        ArgumentList = Expression ("," Expression)*

        Block = "{" Statement* "}"
        Statement = VariableDeclaration | Assignment | MethodCall | PrintStatement |
ConditionalStatement | LoopStatement

        VariableDeclaration = VarType identifier "=" Expression ";"
        VarType = "সংখ্যা" | "হাছামিছা" | "দড়ি" | "বিন্যাস"
        Expression = number | boolean | string | identifier

        Assignment = identifier "=" Expression ";"

        identifier = letter (letter | digit)*
        number = digit+
        boolean = "সত্য" | "মিথ্যা"
        PrintStatement = "দেখাও" "(" string ")" ";"
        string = "\\"" (~"\\"" any)* "\\""

        ConditionalStatement = যদি "(" Expression ")" Block "নয়তো" Block
        LoopStatement = "যতদিন" "(" Expression ")" Block
  }
`;
```

```
const OOP = ohm.grammar(oopGrammar);
const semantics = OOP.createSemantics();

/**
 * Semantic rules for OOP
 */
semantics.addOperation("toTS()", {
  ClassDeclaration(_class, name, _open, variables, constructor, methods, _close) {
        // Handling class variables (properties) using transpileVariables
        const classVariables = variables.children.map(v => transpileVariables(v)).join("\n");

        return `class ${transliterateBangla(name.sourceString)}
{\n${classVariables}\n${constructor.toTS()}${methods.children.map(m =>
m.toTS()).join("\n")}\n}`;
  },

  ConstructorDeclaration(_constructor, _open, params, _close, block) {
        return `constructor(${params ? params.toTS() : ""}) ${block.toTS()}`;
  },

  MethodDeclaration(_method, name, _open, params, _close, block) {
        return `${transliterateBangla(name.sourceString)}(${params ? params.toTS() : ""})
${block.toTS()}`;
  },

  ObjectInstantiation(name, _eq, _new, className, _open, args, _close, _semicolon) {
        return `let ${transliterateBangla(name.sourceString)} = new
${transliterateBangla(className.sourceString)}(${args ? args.toTS() : ""});`;
  },

  MethodCall(object, _dot, method, _open, args, _close, _semicolon) {
        return
`${transliterateBangla(object.sourceString)}.${transliterateBangla(method.sourceString)}(${args
? args.toTS() : ""});`;
  },

  VariableDeclaration(type, name, _eq, value, _semicolon) {
        return transpileVariables(type, name, _eq, value, _semicolon); // Using transpileVariables
  },
```

```
  ParameterList(first, rest) {
        return [transliterateBangla(first.sourceString), ...rest.children.map(p =>
transliterateBangla(p.sourceString))].join(", ");
  },

  ArgumentList(first, rest) {
        return [first.toTS(), ...rest.children.map(a => a.toTS())].join(", ");
  },

  Block(_open, statements, _close) {
        return `{\n${statements.children.map(s => transpileStatements(s)).join("\n")}\n}`;
  },

  PrintStatement(_print, _open, str, _close, _semicolon) {
        return transpileStatements(_print, _open, str, _close, _semicolon); // Using
transpileStatements
  },

  string(_open, chars, _close) {
        return `"${chars.sourceString}"`;
  },

  ConditionalStatement(_if, _openParen, condition, _closeParen, ifBlock, _else, elseBlock) {
        return transpileConditionals(_if, _openParen, condition, _closeParen, ifBlock, _else,
elseBlock);
  },

  LoopStatement(_while, _openParen, condition, _closeParen, block) {
        return transpileLoops(_while, _openParen, condition, _closeParen, block);
  }
});

/**
 * Function to transpile BanglaScript OOP to TypeScript
 * @param code - BanglaScript code
 */
export default function transpileOOP(code: string): string {
  const matchResult = OOP.match(code);
  if (matchResult.failed()) {
```

```
        throw new Error("Syntax Error in OOP statement: " + matchResult.message);
  }
  return semantics(matchResult).toTS();
}
```