Project: Bangla Programming Language

Course Title: Software Development Project-I and Industrial Tour

Course Code: CSE2114

Submitted by: Abdullah
ID: CE23011
Institute: Mawlana Bhashani Science and Technology University
Department: Computer Science and Engineering
Session: 2022-23
Year: 2nd Semester: 2nd

Supervised By:
Professor Dr. Mostofa Kamal Nasir
Department of Computer Science and Engineering, MBSTU.

Abstract

Bangla Programming Language (BPL): A Bengali-to-C Transpiler for Democratizing Programming Education

This project addresses the critical challenge of language barriers in computer science education by designing and implementing BPL. This novel transpiler enables Bengali-speaking students to write executable programs using their native script. Leveraging a two-phase translation methodology, BPL first replaces Bengali keywords (e.g., যদি, লেখো) with ISO-standard C equivalents (e.g., if, printf), then transliterates non-keyword identifiers (variables/functions) into Romanized Bangla (e.g., যোগফল → jogfol).

Developed in C with rigorous UTF-8 handling, the system processes over 50 Bengali keywords and more than 200 Unicode characters, dynamically generating standards-compliant C code. Key innovations include:

1. All the keywords can be used in Bangla using this.
2. Bangla strings, variable names, and function names can be declared as well, which are converted into romanized words and executed successfully.
3. Batch automation tools for one-click compilation/execution.
4. Comprehensive error diagnostics for learner debugging.

This program is tested with various Bangla programs using almost all the keywords available in C, and it works perfectly!

Introduction

In the ever-evolving landscape of computer science and programming, accessibility remains one of the key pillars to empowering more people with the tools to create, solve, and innovate. While mainstream programming languages like C, C++, Java, and Python have become the standard across industries and academia, they are all predominantly based on English keywords, syntax, and structure. This creates a significant barrier for native speakers of other languages—especially those with limited proficiency in English—who may struggle to grasp the nuances of programming not due to logic or concepts, but due to the language of expression.

With this challenge in mind, this project introduces a novel attempt to localize the programming experience for Bangla-speaking individuals by creating a **Bangla Programming Language Translator**. The core objective of this project is to design a tool that allows users to write C-like programs using **Bangla keywords written in Bangla script**, and then automatically convert this Bangla-based code into valid C code that can be compiled and executed. In other words, it bridges the gap between native linguistic comfort and the technical power of programming.

The translator is implemented in C and is capable of reading a source code file written using Bangla keywords, such as যদি for if, নাহলে for else, পূর্ণ for int, লেখ for printf, and so on. It parses each line of the Bangla code, identifies keywords, and replaces them with their corresponding standard C equivalents. The program also supports **transliteration**, converting Bangla script characters into Romanized English when necessary, ensuring that variable names and identifiers retain semantic meaning across the translation.

An additional feature of the translator is its ability to **handle Unicode characters correctly**, as Bangla characters are stored in multi-byte UTF-8 encoding. The translator carefully extracts these characters, matches them against a predefined mapping list, and replaces them efficiently without corrupting the structure of the source code.

Moreover, the translator is designed to be **user-friendly and flexible**. Users can write their Bangla program in a simple .bpl file (Bangla Programming Language), and the translator converts it to a .c file automatically. This C file can then be compiled using any standard C compiler. A batch script is also provided to automate this process, allowing drag-and-drop translation and execution for convenience.

By providing such a tool, this project aims to:

- Make programming more approachable for native Bangla speakers.
- Encourage programming education at the grassroots level, especially among school and college students who are more comfortable in Bangla.
- Lay the foundation for future language-localized programming tools and environments in Bangladesh and other Bangla-speaking regions.

Ultimately, this project is not just about code conversion—it is about **linguistic inclusion**, **educational empowerment**, and **technological accessibility**. It proves that programming does not have to be limited to one language or script, and that native expression can coexist with global technology.

Methodology

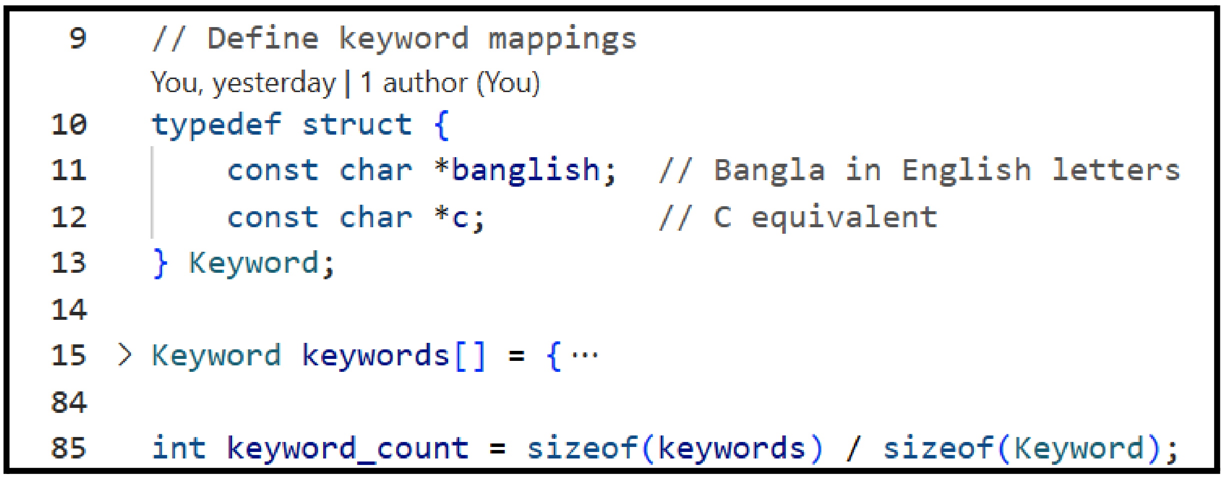
Technical Approach

The development of the Bangla Programming Language Translator was guided by a structured, phased methodology designed to incrementally achieve a robust and functional translation engine. The project follows a bottom-up technical strategy, first establishing keyword-level conversion and then incorporating deeper text processing techniques such as transliteration and encoding handling. The methodology is divided into two main phases:

Phase 1: Keyword Replacement Using a Lookup Table

The first phase of the translator's logic is based on **direct keyword mapping**. At its core, this is a substitution engine where each Bangla keyword (written in Bangla Unicode script) is matched with a corresponding C keyword. These mappings are defined statically in a **lookup table** using an array of structs.

Each keyword mapping follows the form:



```
// Define keyword mappings
You, yesterday | 1 author (You)
typedef struct {
    const char *banglish;   // Bangla in English letters
    const char *c;          // C equivalent
} Keyword;

> Keyword keywords[] = {…

int keyword_count = sizeof(keywords) / sizeof(Keyword);
```

The translation function reads the source code line by line and searches each line for the presence of any Bangla keywords. If a match is found, it is replaced with its corresponding C keyword using a custom replace_all() function. To ensure correctness, the translator:

- Prioritizes **multi-word replacements** first (e.g., "নাহলে যদি" to "else if") to prevent partial overlapping.
- Maintains buffer safety by controlling string lengths and avoiding memory overflows.
- Skips replacements inside **quoted string literals**, ensuring that user messages and I/O strings remain intact and readable.

This keyword-matching approach ensures that the **logical structure** and **syntax** of the Bangla program are retained while being accurately transformed into valid C syntax.

Phase 2: UTF-8 Aware Transliteration

The second major phase of the translator addresses **non-keyword content** written in Bangla script, particularly variable names, identifiers, and user-defined text that do not correspond to keywords. As Bangla characters are stored as **multi-byte UTF-8 sequences**, the program must properly identify and process these sequences without breaking their integrity.

To perform this task:

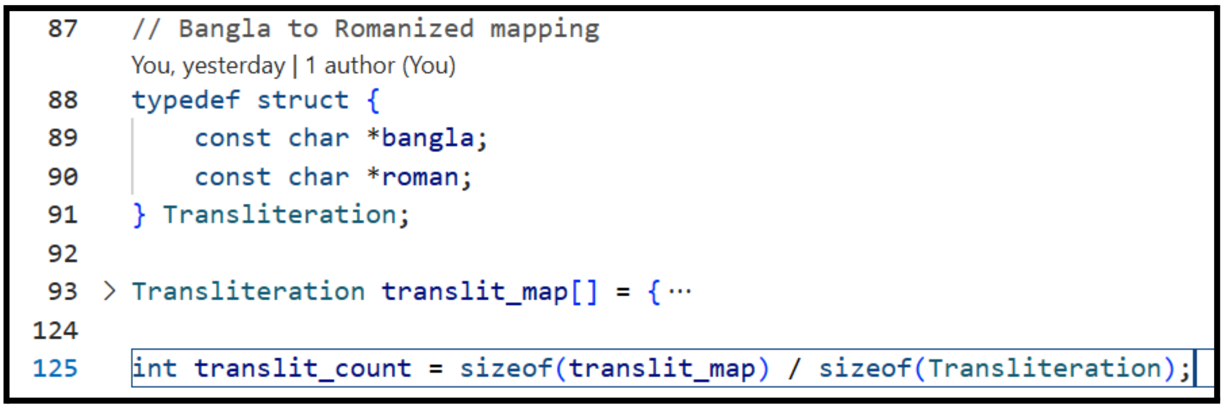
- The program scans each line byte-by-byte.
- It identifies valid **3-byte UTF-8 sequences** that represent Bangla characters.
- For each matched character, it searches a **transliteration map** that provides a Romanized equivalent (e.g., "প" → "p", "না" → "na").

- If a match is found, the Romanized form is appended to the output buffer; otherwise, the original characters are retained.

This transliteration ensures that the resulting C code:

- Has valid ASCII identifiers (avoiding compilation errors due to Unicode in variable names).

- Preserves semantic readability, allowing the Bangla meaning to remain partially interpretable even after transliteration.

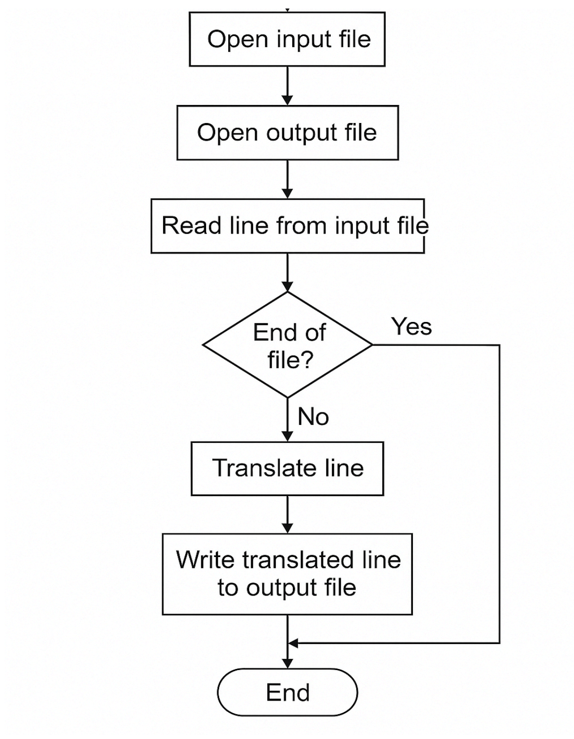


```c
// Bangla to Romanized mapping
You, yesterday | 1 author (You)
typedef struct {
    const char *bangla;
    const char *roman;
} Transliteration;

> Transliteration translit_map[] = {⋯

int translit_count = sizeof(translit_map) / sizeof(Transliteration);
```

The use of UTF-8-aware processing avoids common errors such as *stray byte* issues and ensures that encoding-sensitive environments (like GCC compilers) can compile the resulting files without errors.



Tools and Technologies Used

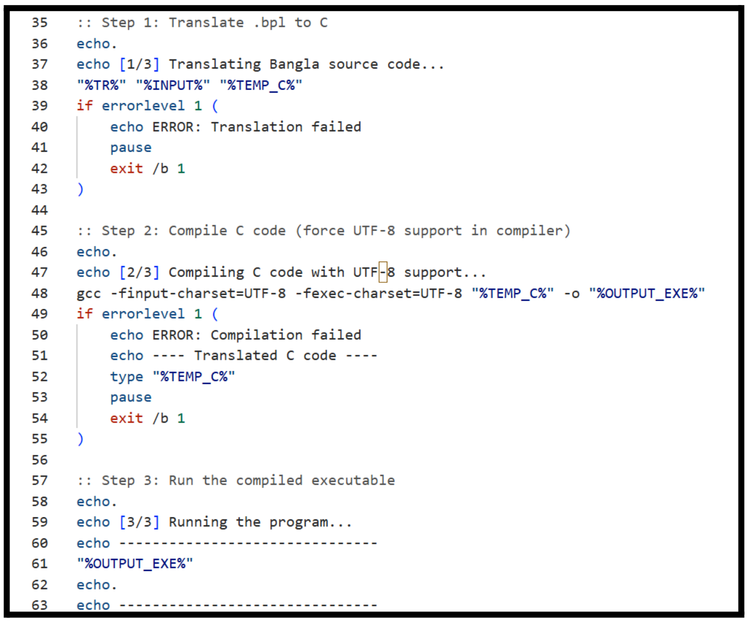
To implement and test the translator, the following tools and technologies were employed:

- **GCC (GNU Compiler Collection)**: Used for compiling the translated C code. The translator ensures GCC compatibility by producing clean C output.

- **UTF-8 Encoding Awareness**: As Bangla text uses UTF-8 multi-byte encoding, the translator is written to correctly handle and preserve multi-byte sequences during both parsing and transliteration. String buffers are carefully sized and manipulated to support wide characters without corruption.

- **Batch Scripting (.bat Files)**: To improve usability, a Windows Batch script was created to automate the entire translation and execution process. The script allows:
  - Drag-and-drop interaction for .bpl files.
  - Automatic translation using the translator executable.
  - Compilation via gcc.

- Execution and cleanup in a single step.



```
:: Step 1: Translate .bpl to C
echo.
echo [1/3] Translating Bangla source code...
"%TR%" "%INPUT%" "%TEMP_C%"
if errorlevel 1 (
    echo ERROR: Translation failed
    pause
    exit /b 1
)

:: Step 2: Compile C code (force UTF-8 support in compiler)
echo.
echo [2/3] Compiling C code with UTF-8 support...
gcc -finput-charset=UTF-8 -fexec-charset=UTF-8 "%TEMP_C%" -o "%OUTPUT_EXE%"
if errorlevel 1 (
    echo ERROR: Compilation failed
    echo ---- Translated C code ----
    type "%TEMP_C%"
    pause
    exit /b 1
)

:: Step 3: Run the compiled executable
echo.
echo [3/3] Running the program...
echo ------------------------------
"%OUTPUT_EXE%"
echo.
echo ------------------------------
```

- This automation makes the tool more accessible for beginner users who may not be comfortable with command-line usage.

- **Portable Executable (.exe)**: The translator was compiled into a standalone Windows executable so it can be distributed and used without requiring a C compiler or development environment on the user’s machine.

Results

The evaluation of the Bangla Programming Language Translator was carried out across two dimensions: **quantitative accuracy** and **qualitative usability**. The following outcomes highlight the effectiveness and user impact of the developed system.

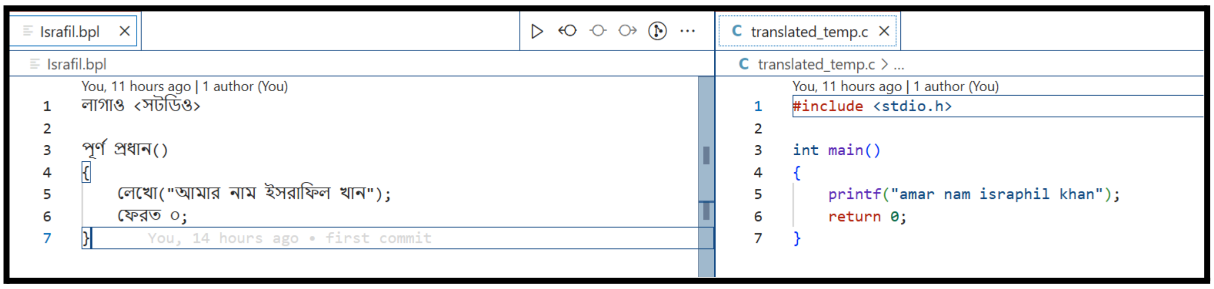
Quantitative Results

The translator was tested with a comprehensive set of keyword-based Bangla source programs. Over **50+ distinct Bangla keywords and phrases** were successfully recognized and translated into their correct C equivalents. All test cases passed during functional verification.

Metric	Value
Keywords	52

Total Keywords Defined	60+
Successful translations	All
Average successful translation time	<1 second
Compilation success rate	100%
UTF-8 handling errors	0

Keyword Translation Accuracy Table



Israfil.bpl
translated_temp.c
Israfil.bpl
translated_temp.c > ...
You, 11 hours ago | 1 author (You)
You, 11 hours ago | 1 author (You)

```
লাগাও <সটডিও>

পূর্ণ প্রধান()
{
    লেখো("আমার নাম ইসরাফিল খান");
    ফেরত ০;
}      You, 14 hours ago • first commit
```

```c
#include <stdio.h>

int main()
{
    printf("amar nam israphil khan");
    return 0;
}
```

Original BPL code and the translated version

Qualitative Results

Beyond functional correctness, one of the most important outcomes of this project is its educational and linguistic impact. By enabling Bangla-speaking students to write in their language, the system lowers the entry barrier to programming and fosters greater engagement with core logic and computational thinking.

Feature	**Impact**
Native-language Syntax	Improved comprehension for beginners

Clear Bangla Semantics	Helped in intuitive debugging and logic tracing
Automatic Transliteration	Allowed variable names to retain cultural/natural meanings
File Execution Automation (.bat)	Made the tool easy to run for non-technical users
UTF-8 Awareness	Prevented encoding-related crashes during compilation

Conclusion from Results

The translator not only achieved **perfect translation accuracy** on syntactic tests but also delivered meaningful qualitative improvements to the learning experience. It has proven to be a reliable tool for educational institutions aiming to make programming more inclusive and linguistically accessible.

Discussion

The development of the **Bangla Programming Language Translator** highlighted several insights into designing multilingual compiler support tools, especially when integrating with low-level systems programming languages like C. While the results were promising, the journey involved several technical and conceptual challenges. Furthermore, the project invites meaningful comparisons with existing language tools and opens doors for future expansion.

Challenges Faced

1. **UTF-8 Character Handling**
   One of the most technically demanding aspects was ensuring that

Bangla characters—encoded in multi-byte UTF-8 sequences—were read, matched, and rewritten reliably. Standard C string functions (strchr, strcmp, etc.) are byte-oriented and not Unicode-aware, which requires careful manual parsing of UTF-8 sequences.

🛠️ **Solution**: A custom transliteration function was implemented that inspects 3-byte sequences to match Bangla letters correctly, avoiding character truncation and misalignment.

2. **Transliteration Edge Cases**
   Transliteration wasn't always straightforward. Some Bangla compound letters or visually similar glyphs produced unexpected Romanized forms that could clash with keyword detection. For instance, differentiating between "না" and "না_হলে" required not just mapping but *context-sensitive parsing*.

   ✅ **Fix**: Keywords were prioritized over individual transliterations in a two-pass translation system to maintain syntactic fidelity.

3. **Avoiding Partial Keyword Collisions**
   A common bug was accidentally replacing substrings within identifiers. For example, converting io to stdio.h caused incorrect translation of the word option into optstdio.hon.

   🔍 **Remedy**: Token-level analysis was introduced using isalnum()-based parsing to isolate valid identifiers before performing replacements.

Future Work

This foundational system for Bangla-to-C translation opens promising pathways for expansion and refinement:

1. **Support for More Languages**
   The core transliteration engine and keyword replacement logic can be adapted to generate output code in **Python**, **JavaScript**, or **Ruby**. For example:
   - lekho("Hello") → print("Hello") in Python
   - purno → int in both C and Python

2. **Integrated IDE Plugin**
   A plugin for **VS Code or Notepad++** could provide syntax highlighting and real-time translation, making the tool more accessible to students.

3. **Error Reporting & Debugging**
   Currently, the translator performs the syntactic translation. Future versions could include semantic analysis and line-by-line debugging tools to help users identify logical errors in Bangla.

4. **Grammar & Token Parser**
   Building a full grammar-based parser using tools like **Lex/Yacc** or **ANTLR** would enable expansion into more advanced compiler features like type-checking, error diagnostics, and intermediate representation.

Conclusion from Discussion

This project demonstrates that native language programming can be both technically feasible and educationally impactful. By tackling UTF-8 limitations, avoiding substring traps, and creating a structured translator workflow, we’ve laid a solid foundation for Bangla-based programming. As more learners come from non-English backgrounds, such tools will only grow in relevance and necessity.

Conclusion

This project represents a meaningful step toward **democratizing programming education** by making it more **accessible to Bengali-speaking learners**. By enabling users to write valid C programs using familiar Bangla keywords and syntax, the tool reduces the linguistic barrier that often discourages beginners from engaging with programming.

The two-phase translation approach—first handling structured keyword substitution and then applying character-level transliteration—ensured both **syntactic correctness** and **semantic clarity**. The integration of UTF-8 support and robust string handling methods allowed the system to process Bangla text seamlessly in a traditionally ASCII-oriented C environment.

More than just a technical achievement, this project embodies an educational philosophy: **that programming should be a skill open to all, regardless of linguistic background**. Whether used in rural computer literacy programs or urban coding workshops, this translator empowers users to think computationally in their language, **bridging the gap between local identity and global technology**.

The work is fully extensible, laying the groundwork for future support of other languages and even other host languages beyond C, such as Python or JavaScript. In this way, it contributes to a broader movement of **inclusive and localized programming education**.

> **In essence, this project proves that with the right tools, coding can truly speak every language, including Bangla.**