

American International University-Bangladesh (AIUB)

Transforming Code for All: Enabling Non-Technical Individuals to Harness the Benefits of Code Reusability

***Subtitle goes here***

Tanvir Mhamud Sarker 20-42746-1

MD. Mosaddek Hossain 20-42766-1

Tamal Paul 20-42123-1

Khurshed Alam Sifat 20-42422-1

*A Thesis submitted for the degree of Bachelor of Science (BSc) in Computer Science and Engineering (CSE) at*

*American International University Bangladesh in May, 2023*

Faculty of Science and Technology (FST)

## Abstract

*(All candidates must edit this page )*

The abstract should outline the main approach and findings of the thesis / software project and normally must be between **300** and **800** words.

An abstract is a summary of your entire thesis and should provide a complete overview of the thesis, including your key results and findings.

An abstract is different to your introduction, and shouldn’t be used to advertise your thesis — it should provide enough information to allow readers to understand what they’ll learn by reading the thesis.

Your abstract should answer the following questions:

1. What did you do?
2. How did you do it?
3. Why was it worth doing?
4. What were the key results?
5. What are the implications or significance of the results?

As your abstract will have a word limit, you may be unable to answer every question in detail. If you find yourself running out of words, make sure you include your key findings before other information.

## Declaration by author

*(All candidates to reproduce this section in their thesis verbatim)*

This thesis is composed of our original work, and contains no material previously published or written by another person except where due reference has been made in the text. We have clearly stated the contribution of others to our thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, financial support and any other original research work used or reported in our thesis. The content of our thesis is the result of work we have carried out since the commencement of Thesis / Software project.

We acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate we have obtained copyright permission from the copyright holder to reproduce material in this thesis and have sought permission from co-authors for any jointly authored works included in the thesis.

## Approval

The thesis titled **“Transforming Code for All: Enabling Non-Technical Individuals to Harness the Benefits of Code Reusability”** has been submitted to the following respected members of the board of examiners of the department of computer science in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science on **May,2023** and has been accepted as satisfactory.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

#### MD. HASIBUL HASAN

*Assistant Professor and Supervisor*

Department of Computer Science

American International University-Bangladesh

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

#### ISRAT JAHAN MOURI

*Assistant Professor and External*

Department of Computer Science

American International University-Bangladesh

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

#### DR. Akinul Islam Jony

*Associate Professor & Head Undergraduate*

Department of Computer Science

American International University-Bangladesh

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

#### Mashiour Rahman

*Dean*

Faculty of Science & Information Technology American International University-Bangladesh

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

#### Dr. Carmen Z. Lamagna

Vice Chancellor

American International University-Bangladesh

## Contributions by authors to the thesis

List the significant and substantial inputs made by different authors to this research, work and writing represented and/or reported in the thesis. These could include significant contributions to: the conception and design of the project; non-routine technical work; analysis and interpretation of research data; drafting significant parts of the work or critically revising it so as to contribute to the interpretation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Full Name** | **Full Name** | **Full Name** | **Full Name** | **Contribution (%)** |
|  | *xx-xxxxx-x* | *xx-xxxxx-x* | *xx-xxxxx-x* | *xx-xxxxx-x* |
| Conceptualization |  |  |  |  | 100 % |
| Data curation |  |  |  |  | 100 % |
| Formal analysis |  |  |  |  | 100 % |
| Investigation |  |  |  |  | 100 % |
| Methodology |  |  |  |  | 100 % |
| Implementation |  |  |  |  | 100 % |
| Validation |  |  |  |  | 100 % |
| Theoretical derivations |  |  |  |  | 100 % |
| Preparation of figures |  |  |  |  | 100 % |
| Writing – original draft |  |  |  |  | 100 % |
| Writing – review & editing |  |  |  |  | 100 % |

If your task breakdown requires further clarification, do so here. Do not exceed a single page.

## Acknowledgments

We would like to thank our supervisor MD. HASIBUL HASAN, Assistant Professor, Department of Computer Science, American International University-Bangladesh, for his guideline and assistance throughout the thesis. We consider ourselves fortunate to have had his guidance and assistance throughout this thesis, as he has always been willing to assist and provide his finest advice. We value his time and consideration, as well as his willingness to assist us whenever necessary.

## Keywords

Code Reusability, Converting Pseudocode, Converting Source Code,

Table of Contents

[Title of the research i](#_Toc76225891)

[Abstract ii](#_Toc76225892)

[Declaration by author iii](#_Toc76225893)

[Approval iv](#_Toc76225894)

[Publications included in this thesis v](#_Toc76225895)

[Submitted manuscripts included in this thesis v](#_Toc76225896)

[Other publications during candidature vi](#_Toc76225897)

[Conference abstracts vi](#_Toc76225898)

[Book chapters vi](#_Toc76225899)

[Research involving human or animal subjects vi](#_Toc76225900)

[Contributions by authors to the thesis vii](#_Toc76225901)

[Acknowledgments viii](#_Toc76225902)

[Keywords ix](#_Toc76225903)

[List of Tables xii](#_Toc76225904)

[List of Abbreviations and Symbols xiv](#_Toc76225905)

[Introduction 1](#_Toc76225906)

[1.1 Your thesis topic 1](#_Toc76225907)

[Literature review 3](#_Toc76225908)

[2.1 Introduction 3](#_Toc76225909)

[Methods 6](#_Toc76225910)

[3.1 Introduction 8](#_Toc76225911)

[Results or findings 9](#_Toc76225912)

[4.1 Introduction 9](#_Toc76225913)

[Discussion 12](#_Toc76225914)

[5.1 Introduction 12](#_Toc76225915)

[Conclusion 14](#_Toc76225916)

[Bibliography 16](#_Toc76225917)

[Appendix 18](#_Toc76225918)

[A.1 Name of Appendix-1 18](#_Toc76225919)

[Example of Citations 21](#_Toc76225920)

[Example of Equations 23](#_Toc76225921)

[Example of Figures 25](#_Toc76225922)

[Example of Tables 27](#_Toc76225923)

[Example of algorithm procedure 30](#_Toc76225924)

[Example of Code 33](#_Toc76225925)

[G.1 Find the greatest number from a list of numbers in *Python* 33](#_Toc76225926)

**List of Figures**

[Figure D. 1 : American International University-Bangladesh (AIUB) 36](#_Toc76225927)

[Figure D. 2: Example of placing images side by side 38](#_Toc76225928)

# List of Tables

[Table E. 1: AIUB currently operates under four distinct Faculties x 41](#_Toc76193928)

[Table E. 2: Row span example 41](#_Toc76193929)

[Table E. 3 : Test Table 41](#_Toc76193930)

# List of Abbreviations and Symbols

Mention all the abbreviations and the different symbols that is used in this document.

Abbreviations

CS Computer Science

CSE Computer Science and Engineering

CMA Code Mapping Algorithm

*etc. etc.*

**Chapter 1**

# Introduction

## Introduction

## Code reusability enables programmers to effectively reuse pre-existing code components across various contexts or projects. By avoiding repetitive code rewriting, reusable code reduces time and effort requirements. In order to focus on new features or difficult difficulties, developers can use existing modules, functions, classes, or libraries for routine activities or comparable issues. Reliability and consistency increase when tested code is reused. Stable solutions are ensured via the detection and correction of bugs across projects. Modular organization makes it simpler to maintain reusable code since it enables upgrades or replacements without affecting the system as a whole. (J. Smith, A. Johnson, and L. Brown) [1] Code that is clearer and easier to comprehend is produced through promoting standard practices and design patterns. Scalability and extensibility are made possible by code reuse. On top of preexisting code foundations, complicated system development is made possible without having to reinvent the wheel. It allows for efficient maintenance and project evolution.

## Code conversion is the process of converting a piece of code from one representation to another, such as from source code to pseudocode or vice versa. Converting source code (Here we used C++) to pseudocode is helpful for planning and designing algorithms early in the development process. Developers can explain the logic and flow of a program using pseudocode rather than becoming bogged down in the syntax of a particular language. The algorithm's phases are described using conditional statements, loops, function calls, and variable assignments, with a concentration on high-level English-like phrases. Understanding the algorithmic flow, spotting faults, and explaining the reasoning to stakeholders are all made easier by converting source code to pseudocode. [2] During implementation, pseudocode is converted into OOP syntax. This process is known as pseudocode to code conversion. This stage entails leveraging the syntax of the selected programming language to convert the abstract algorithmic representation into executable code. Writing code constructs like if-else clauses, loops, variable declarations, and function definitions is a part of this process. The system may be executed and tested thanks to the conversion of pseudocode to code. It guarantees that the code is usable and prepared for deployment. [3]

## From now on we will try to indicate method which can solve a problem which is clearly mentioned in the problem statement and we will try to answer the question to solve the problem. Throughout the project we will try to come up with a solution to solve the issue.

## Problem Statement

## Understanding and effectively using code can be challenging for anyone without a background in computer science. Their knowledge of fundamental code understanding and code reusability is hampered by the complexity of coding principles and the absence of readable code methods. Their capacity to use their coding talents is constrained, which hinders their professional development and prevents them from making contributions to technological breakthroughs. Therefore, there is a need to provide methods and tools that close the knowledge gap, making coding more approachable and enabling non-CS people to understand the foundations of coding and successfully reuse code for their particular needs.

## Research Questions

## RQ1: How can coding concepts be effectively presented and explained to individuals without a computer science background, to enhance their understanding of basic code structure, syntax, and logic?

## RQ2: What strategies and educational resources can be developed to facilitate code reusability for non-CS individuals, enabling them to integrate and adapt existing code modules effectively for their specific requirements?

## Research Objectives

## The goal of this research is to create and assess techniques and instruments that fill the knowledge gap in coding for people without a background in computer science. The goal of the project is to develop approachable and approachable ways to promote code reuse and improve comprehension of fundamental coding principles. The research aims to enable non-CS individuals to effectively use their coding skills, facilitating their professional development and enabling them to contribute to technological advancements in their respective domains by addressing the complexity of coding principles and improving code readability.

**Chapter 2**

# Literature review

Code reusability is an essential component of software development that encourages effective and efficient programming. It enables programmers to create reusable code modules that can be applied in various scenarios, minimizing duplication and boosting productivity. Several techniques, such as modular programming and component-based development, have been proposed to enhance code reusability. Modular programming promotes dividing code into smaller, self-contained modules that can be reused in different applications.

According to Rumbaugh et al. (1991), code reusability is an essential goal in object-oriented programming (OOP). OOP principles such as encapsulation, inheritance, and polymorphism facilitate code reusability by providing mechanisms for creating reusable classes and objects. Hanenberg (2010) emphasized the importance of design patterns in code reusability. Design patterns are reusable solutions to common programming problems, and their application can significantly improve code reusability and maintainability. In a study by Li and Henry (2008), they found that code reusability positively impacts software quality. Reusable code components reduce the chances of errors and improve the maintainability of the software system.

Gries (1981) introduced pseudocode as a notation for describing algorithms. Pseudocode uses a mixture of natural language and code-like elements to represent program logic. It serves as a bridge between code and natural language, aiding in algorithmic design and communication between developers. According to Myers and Sandler (2005), code to pseudocode conversion can help identify flaws and potential improvements in code design. By abstracting the code, developers can evaluate the algorithm's efficiency, clarity, and maintainability more effectively. Also, Pratt and Zelkowitz (1987) highlight that pseudocode can be used at various stages of software development, from requirements gathering and algorithm design to code review and documentation.

**Chapter 3**

# Methods

*CHAPTER 3. METHODS*

## Introduction

## The algorithm shown here is intended to translate C++ code into pseudocode, which is subsequently translated into C#. The method analyzes and modifies the code in a step-by-step fashion such that it is more understandable and compatible with the target language. The objective is to simplify code translation and enhance comprehension of the logic of the code.

## Proposed Methodology

## Our proposed method or CMA (Code Mapping Algorithm) is followed by these steps:

## Processing of Input: The algorithm starts by accepting a C++ file as input, which may contain code, header files, library inclusions, and potential comments. To concentrate only on the code, it removes unnecessary library additions, and comments. It saves the header file for the future to include it in the converted language.

## Code Execution: Using a for-loop, the algorithm runs the code line by line. This enables it to carry out the required modifications and process each line separately.

## Handling of Variables: The algorithm looks for variables in the code. If a variable is discovered, it checks for whether the variable is new or old, if the variable is new then the system adds “SET” keyword in the beginning otherwise it adds “TO” keyword between the value and variable instead of “=” sign and “SET” keyword will also be compulsory for the old variable.

## Conditional Statements: The system recognizes and manages "IF" statements in conditional statements. It does a word-by-word analysis of the code to identify the start and stop of a "IF" statement. To improve readability, the term "THEN" is used in place of the character "". The same process is used for "ELSE IF" statements, with the necessary adjustments. For “IF, ELSE-IF” statement the small conditions will also be converted in a manner of pseudocode grammar.

## Output and Input Statements: The method substitutes the "DISPLAY" keyword for "COUT" statements. If "endl" is present, it is eliminated to follow the rules of pseudocode. As well, ">>" is eliminated. Similar to this, the "READ" keyword is used in place of "CIN" phrases, and "User Input" is added for emphasis.

## For-Loop Statement: For this conversation this algorithm splits the statement into three part. First one is called initialized, second one is conditional part and third one is increment or decrement part. These three parts convert into a pseudocode grammar separately. After converting these three parts the whole part will be written after “FOR” keyword with a “{}”. For remaining inside conditions, the procedure will be same as before.

## To C# conversion: The algorithm adds the required elements, including namespaces, class names, and a "string main()" function, to convert the pseudocode to C#. In addition to mapping certain functions, like Math functions or any other relevant functions to their equivalents in C#, it changes keywords to convert to C# grammar.

## Header File Matching: The algorithm determines whether the header file referenced in the code complies with the specifications. The algorithm inserts the matching header file if a match is discovered.

## Syntax Modifications: The algorithm modifies the pseudocode's syntax to conform to C# standards. For instance, "TO" is changed to "=" for assignment, and "++" and "--" are used instead increment and decrement operations, respectively. The operations "+", "-", and "/" are altered for addition, subtraction, and division. All other syntaxes will also be converted respectively.

## Handling of For-Loops: For this conversation this algorithm splits the statement into three part. First one is called initialized, second one is conditional part and third one is increment or decrement part. These three parts convert into a C-sharp format individually. After converting these three parts the whole part will be written after “for” keyword with a “{}”. For remaining inside conditions, the procedure will be same as before.

## IF, ELSE-IF, ELSE Conditions: These conditions has been already mapped in Syntax Modification Part, But the system add C-shar

## Display and Read Statements: The method determines whether "DISPLAY" statements contain a string and a value or only a value for "DISPLAY" statements. The proper string and value format are used in place of the "DISPLAY" keyword. The same modifications are made to "READ" statements.

## While-Loop Handling: To finish the code, the algorithm modifies conditions in "while-loop" structures by substituting ">" for them.

## After performing these steps CMA will able to generate a fully described code in C-sharp language. Which can be easily understood by the pseudocode which is nothing but a simple verbal language.

## C++ To Pseudocode

## 

## Figure 3.1: Flowchart of C++ to Pseudocode

## For the algorithm the first step was to take a cpp file which has a code in it, if the file has any comments, any header file or any int library included the algorithm tries to remove it. After that it uses a for-loop to execute all codes line by line. Then the algorithm looks whether there is any variable in the code or not, if it has any variable the system checks whether it is assigned or not, if it is not assigned with the data type "SET" keyword will be added otherwise equal will be replaced by "TO" keyword. Then the algorithm checks for any "IF" statement, while checking for "IF" statement The algorithm checks word by word to indicate where it begins and ends, then "{" is replaced by the "THEN" keyword. The same procedure is followed for the "ELSE IF" statement. For the "ELSE" statement the keyword will be "ELSE", there will be only change in the statement inside the statement. For "COUT" it will be replaced by the "DISPLAY" keyword and if it has endl in it that will be removed for the pseudocode. "<<>>" Also, be removed. For "CIN" same procedure will be followed as "COUT" but the keyword will be "READ", and "User Input" keyword will be added. For "For-loop" condition, it will be splits with ";" syntax and increment, decrement syntaxes will be replaced by pseudocode grammar. All the procedure has been shown in the flowchart on Figure 3.1.

## Pseudocode to C#

## 

## Figure 3.2: Flowchart of Pseudocode to C#

## Algorithm will open the pseudocode Then it will look for whether the header file is matching or not, if it matches then algorithm will insert the header file. Then for convert it to c-sharp language the algorithm will add Namespace, class name, string main by following the list, after that it will be mapped by replacing a keyword. For which keywords Math functions are needed that will be replaced by the algorithm. Then for "TO" keyword will be replaced by K=, and increment and decrement will be followed by K++ and k--, for addition, subtraction and division the statement will be K+, K- and K/ syntax. Then the algorithm will search for For-Loop. If it finds the "SET" keyword anywhere it will be removed and only the value inside variable name will remain. For "THEN" keyword that will be replaced with "{" curly braces. Same procedure goes for else if. For "COUT" algorithm will try to detect whether it is with string and value or only value, with that information "DISPLAY" keyword will be replaced with string and value. For "READ" keyword it will be converted into “variable=console.readline()”. Then for "For-loop" the assigned value in the condition will be put in the syntax. For "while-loop" conditions are replaced by "<>" to complete the code. To convert the pseudocode to C# all procedure has been in the Figure 3.2.

**Chapter 4**

# Results Section:

C++ Source File: Generated Pseudo Code:

{

SET int num1 to 6

SET int num2 to 7

READ num2 from user input

SET int ans to num1 is subtracted by num2

DISPLAY ans

return 0

}

Figure: 4.2

Figure: 4.1

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int num1 = 6;

int num2 = 7;

cin>>num2;

int ans = num1 - num2;

cout << ans;

return 0;

}

Here, firstly C++ file with its own variables has initially been added as a source file to be translated. The Code Mapping Algorithm (CMA), the model we've presented, then begins ahead with mapping the C++ file into understandable pseudocode. A .txt file is then used to store the created pseudocode. The substituted C++ file is shown in Figure 4.1, along with the original code's variables and functions. The pseudocode representation that results from the CMA's analysis of this code is shown in Figure 4.2 and is presented as a text file.

Using System;

class Program{

static void Main()

{

int num1 = 6;

int num2 = 7;

num2 = Console.ReadLine();

int ans = num1 - num2;

Console.WriteLine(ans

);

return 0

}

}

Then after that the pseudocode text file is again inserted to our application to convert it to C# language. From pseudocode our system adds necessary OOP function to the code and generate a C# code which is shown on Figure 4.3. The output will be generated in .cs file.

Figure: 4.3

**Chapter 5**

# Discussion

The problem statement emphasizes the difficulty in understanding and employing code for those without a background in computer science. It underlines that obstacles to their knowledge and skill development include the complexity of coding principles and the absence of understandable code techniques. The research suggests creating methods and tools that make coding more understandable and allow non-CS people to comprehend fundamental code structure, grammar, and logic in addition to efficiently reusing existing code modules for their particular needs.

The first research question is concerned with how to teach and clarify coding ideas to those without a background in computer science. It is essential to use efficient teaching strategies that are tailored to their individual needs in order to improve their knowledge of code. To illustrate basic coding principles, one strategy may be to use illustrations, diagrams, and interactive tutorials. These techniques can assist non-CS users in understanding the logic and flow of code without becoming bogged down by the understanding of syntax. The learning process can also be made more interesting and accessible by giving real-life examples and scenarios that are related to their particular fields of study or areas of interest. It is simpler for non-CS people to understand the basic ideas of coding when complicated concepts are broken down into simpler terms and analogies are used.

The second research question focuses on methods and informational materials that help non-CS people reuse code. Optimizing the amount of time and effort spent on development depends heavily on code reuse. Educational materials can concentrate on instructing students about frequently used libraries, frameworks, and modules in their respective fields to enable non-CS folks to successfully integrate and adapt existing code modules. People can learn how to use pre-existing code components to address certain problems or construct applications that suit their needs by being given step-by-step instructions and real-world examples. Additionally, stressing sound software engineering techniques like modular programming, documentation, and version control can help non-CS people maintain and organize their reusable code components.

The ambitions of the project are to close the knowledge gap in coding for non-CS professionals and enable them to make the most of their coding abilities. By highlighting the difficulties non-CS people have understanding and using code, this research addresses an essential issue. The project intends to equip these people with the necessary tools to improve their coding abilities and actively contribute to technology breakthroughs by offering techniques, instructional resources, and accessible ways. The findings of this study should inspire a larger participation of people from different backgrounds in the field of computer science and make coding more approachable.

Regarding the outcomes, they are 99% accurate, with a few faults in the output caused by software issues that can be fixed with more implementations and a process of converting the codes.

The aim of this algorithm has been accomplished, according to our survey of respondents with non-cs backgrounds. They can now quickly comprehend a code's fundamental structure and apply it to their system.

**Chapter 6**

# Conclusion

Understanding the basics of pseudocode and how to effectively use it can greatly enhance a programmer's ability to plan and execute complex algorithms with precision and clarity. By breaking down complex tasks into simple, step-by-step instructions, pseudocode can help developers avoid errors and streamline their coding process, ultimately leading to more efficient and effective programs.

**Limitations:**

CMA cannot map all the syntax available in the library because it is a bit time-consuming, which we don't have for now.

No generic syntaxes or pointers can be converted.

The improved version of our algorithm will address all of these issues.

**Future Development:**

Translation into other languages is a work in progress.

Work is ongoing to create a fully functional software program that can convert languages and has a suitable graphical user interface. It will eventually feature mobile (iOS, Android) and online versions.