VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**NGUYEN QUOC THANG – 523H0094**

**NGO CHI THUAN – 523H0102**

**MID-TERM ESSAY**

**DISCRETE STRUCTURES**

**HO CHI MINH CITY, 2025**

VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**NGUYEN QUOC THANG – 523H0094**

**NGO CHI THUAN – 523H0102**

**MID-TERM ESSAY**

**DISCRETE STRUCTURES**

Advised by

**M.S Nguyen Quoc Binh**

**HO CHI MINH CITY, 2025**

**ACKNOWLEDGEMENT**

We sincerely express our gratitude to Ton Duc Thang University for providing us with a conductive learning and research environments. Especially thanks to the Faculty of Information Technology and the Software Engineering Department for organizing our study program and we would like to express our gratitude to Master Nguyen Quoc Binh for teaching, sharing knowledge and providing valuable experiences to us during our learning process. We also appreciate her guidance in addressing our inquiries while working on our reports.

*Ho Chi Minh city, 30th April 2025.*

*Author*

*(Signature and full name)*

***Thang***

Nguyen Quoc Thang

***Thuan***

Ngo Chi Thuan

**DECLARATION OF AUTHORSHIP**

We hereby declare that this is our own project and is guided by Mr. Nguyen Quoc Binh; The content research and results contained herein are central and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the main author from different sources, which are clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

**If something wrong happens, we will take full responsibility for the content of my project.** Ton Duc Thang University is not related to the infringing rights, the copyrights that We give during the implementation process (if any).

*Ho Chi Minh city, 30th April 2025*

*Author*

*(Signature and full name)*

***Thang***

Nguyen Quoc Thang

***Thuan***

Ngo Chi Thuan

**TABLE OF CONTENT**

CHAPTER 1. THE TASKS OF EACH MEMBER 1

CHAPTER 2. TRUTH TABLE 1

2.1 Theory of Basic logic 1

2.1.1 Negation 1

2.1.2 Conjunction 2

2.1.3 Disjunction 2

2.1.4 Implication 2

2.1.5 Logical Equivalence 3

2.2 Explain the testcases 3

2.3 Theory of Reverse Polish 6

2.4 Explanation the implementation 6

2.4.1 Explanation for Infix2Postfix(Infix) function 7

2.4.2 Explanation for Postfix2Truthtable(Postfix) function 7

CHAPTER 3. QUANTIFIED REASONING OVER REAL-WORLD DATA USING PREDICATE LOGIC 10

3.1 Dataset 10

3.1.1 Output 11

CHAPTER 4. RSA CRYPTOSYSTEM 11

4.1 RSA Introduction 11

4.2 Implementation 11

4.3 Discussion 16

4.4 Recommend 16

REFERENCES 17

# THE TASKS OF EACH MEMBER

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Date** | **Activities / Tasks / Items** | **Priority** | **Assigned to** |
| 6 |  | Task 1 - Implement code | High | Thuan |
| 4 |  | Task 2 - Implement code | High | Thuan |
| 7 |  | Task 3 - Implement code | High | Thang |
| 1 |  | Search for theoretical documents | High | Thuan |
| 2 |  | Collect images for document | Low | Thang |
| 3 |  | Create dataset for task 2 | High | Thang |
| 8 |  | Find and write references | Low | Thuan |
| 5 |  | Explain task 1 code | High | Thang |
| 9 |  | Explain task 3 code | High | Thang |
| 10 |  | Task 3 - Discussion | High | Thang, Thuan |
| 11 |  | Task 3 - Recommendation | Medium | Thang, Thuan |
| 12 |  | Assign tasks to each member | Medium | Thuan, Thang |
| 13 |  | Task1 - Calculate and verify the testcases | High | Thang |
| 14 |  | Write the acknowledgements section | Medium | Thang |

# TRUTH TABLE

## Theory of Basic logic

### Negation

Definition: If is a statement variable, the **negation** of *p* is “not *p*” or “it is not the case that *p*” and is denoted.

Truth table:

|  |  |
| --- | --- |
|  |  |
| F | T |
| T | F |

### Conjunction

Definition: If and are statement variables, the **conjunction** of and is “*p* and *q*”, denoted .

Truth table:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| F | F | F |
| F | T | F |
| T | F | F |
| T | T | T |

### Disjunction

Definition: If and are statement variables, the **disjunction** of and is “*p* or *q*”, denoted .

Truth table:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| F | F | F |
| F | T | T |
| T | F | T |
| T | T | T |

### Implication

Definition: If and are statement variables, the **conditional** of *q* by *p* is “if *p* then *q*” or “*p* implies *q*”, denoted .

It is false when is true and is false; otherwise, it is true. We called the hypothesis (or antecedent) of the conditional and *q* the conclusion (or consequent).

Truth table:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| F | F | T |
| F | T | T |
| T | F | F |
| T | T | T |

### Logical Equivalence

Definition: Two statement forms are called **logically equivalent** if, and only if, they have identical truth values for each possible substitution of statements for their statement variables. The logical equivalence of statement forms *P* and *Q* is denoted by .

Example:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| F | F | F | F |
| F | T | F | F |
| T | F | F | F |
| T | T | T | T |

and always have the same truth values, hence they are logically equivalent.

## Explain the testcases

Test case 1: R|(P&Q)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| F | F | F | F | F |
| F | F | T | F | T |
| F | T | F | F | F |
| F | T | T | F | T |
| T | F | F | F | F |
| T | F | T | F | T |
| T | T | F | T | T |
| T | T | T | T | T |

Test case 2: ~P|(Q&R)>R

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| F | F | F | T | F | T | F |
| F | F | T | T | F | T | T |
| F | T | F | T | F | T | F |
| F | T | T | T | T | T | T |
| T | F | F | F | F | F | T |
| T | F | T | F | F | F | T |
| T | T | F | F | F | F | T |
| T | T | T | F | T | T | T |

Test case 3: P|(R&Q)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| F | F | F | F | F |
| F | F | T | F | F |
| F | T | F | F | F |
| F | T | T | T | T |
| T | F | F | F | T |
| T | F | T | F | T |
| T | T | F | F | T |
| T | T | T | T | T |

Test case 4: (P>Q) &(Q>R)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| F | F | F | T | T | T |
| F | F | T | T | T | T |
| F | T | F | T | F | F |
| F | T | T | T | T | T |
| T | F | F | F | T | F |
| T | F | T | F | T | F |
| T | T | F | T | F | F |
| T | T | T | T | T | T |

Test case 5: (P|~Q)>~P=(P|(~Q))>~P

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| F | F | T | T | T | T |
| F | T | F | T | T | T |
| T | F | T | F | F | T |
| T | T | T | F | F | T |

Requirements:

* Write **Infix2Postfix(Infix)** function
* Write **Postfix2Truthtable(Postfix)** function

Calculate:

1. R|(P&Q)
2. ~P|(Q&R)>R
3. P|(R&Q)
4. (P>Q)&(Q>R)
5. (P|~Q)>~P=(P|(~Q))>~P

## Theory of Reverse Polish

**Reverse Polish notation** (**RPN**), also known as **reverse Łukasiewicz notation**, **Polish postfix notation** or simply **postfix notation**, is a mathematical notation in which operators *follow* their operands, in contrast to prefix or Polish notation (PN), in which operators *precede* their operands. The notation does not need any parentheses for as long as each operator has a fixed number of operands.

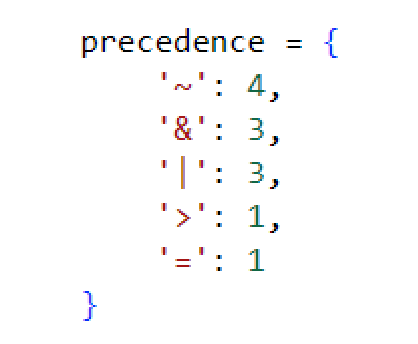
In reverse Polish notation, the operators follow their operands. For example, to add 3 and 4 together, the expression is 3 4 + rather than 3 + 4. The conventional notation expression 3 − 4 + 5 becomes 3 (enter) 4 − 5 + in reverse Polish notation: 4 is first subtracted from 3, then 5 is added to it.

## Explanation the implementation

### Explanation for Infix2Postfix(Infix) function

Input: Infix (A string containing the infix logical expression)

Input data type: str (string).



To begin with, we declare the precedence of logical operators using a dictionary, with decreasing precedence. NOT (~) has the highest precedence, and the lowest is IF AND ONLY IF (=) and IMPLIES (>).

Next, initialize 2 lists, stack (containing mathematical expressions), and Postfix to store returned output.

Then, perform a loop through each character in the input string:

* If it is an operand, add it to Postfix
* If it is an open parenthesis, add it to the Stack.
* If it is a closing parenthesis, take all the operators from the stack and put them into postfix until you reach the opening parenthesis. Remove the opening parenthesis.
* Otherwise, compare operator precedence and put it in postfix

After that, get the remaining operators in the stack.

Finally, Return the postfix string by concatenating the elements in the Postfix list.

### Explanation for Postfix2Truthtable(Postfix) function

Input: Postfix (A string containing the postfix logical expression)

Input data type: str (string).

Keep in mind that Postfix can be got via **Infix2Postfix(Infix**) function.

To begin with, we need to get variables from the postfix expression (Postfix parameter).

variables = sorted(set(filter(str.isalpha, Postfix)))

Where:

* filter(str.isalpha, Postfix): Filter out alphabetic characters (logical variables) from a postfix expression.
* [set()](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Transform​ list to set for the purpose of removing duplicate variables
* [sorted()](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Sort variables in alphabetical order.
* num\_vars: Count the number of logic variables.

Generate all truth value combinations

truth\_combinations = list(product([False, True], repeat=num\_vars))

Where:

* product([False, True], repeat=num\_vars): Generates all possible combinations values (True/False) ​​for logical variables.
* [list()](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Convert results to list.

A close up of text

AI-generated content may be incorrect.Definition of local functions of logical operations:

A screen shot of a computer code

AI-generated content may be incorrect.Use a dictionary to map logical operators to functions.

A black and blue text

AI-generated content may be incorrect.Print table header:

A black text on a white background

AI-generated content may be incorrect.Iterate through each combination of truth values

Where:

* [combination](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): A combination of truth values ​​(e.g. (False, True)).
* zip(variables, combination): Match each variable with its corresponding truth value.
* [dict()](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Creates a dictionary mapping variables to truth values.

A screen shot of a computer code

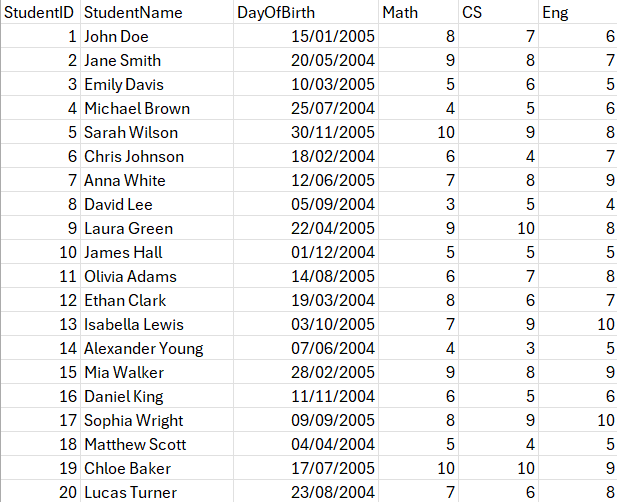
AI-generated content may be incorrect.Calculate the result based on the postfix expression:

Where:

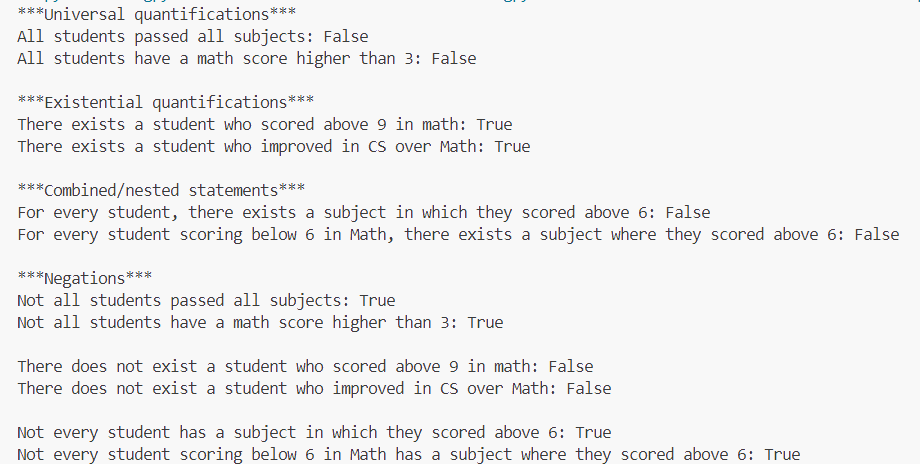
* [stack](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Stack to compute the value of postfix expression.
* [char.isalpha()](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Check if all the characters in the text are letters. In this case, this is used to check if an element in the postfix expression​ is a variable (e.g. P, Q, R, …)
* char in operations: If the character is an operator, perform the corresponding operation by calling the function in operations.

# QUANTIFIED REASONING OVER REAL-WORLD DATA USING PREDICATE LOGIC

## Dataset

We created “student.csv” dataset contains these fields: StudentID, StudentName, DayOfBirth, Math, CS, Eng.

## Output



# RSA CRYPTOSYSTEM

## RSA Introduction

In cryptography, RSA is a public-key cryptography algorithm. It was the first algorithm suitable for generating digital signatures as well as encryption. It marked a major advance in the field of cryptography in the use of public keys. RSA is widely used in electronic commerce and is considered secure provided the key length is sufficiently large.

In a public-key cryptosystem, the encryption key is public and distinct from the decryption key, which is kept secret (private). An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be decrypted by someone who knows the private key.

## Implementation

Import libraries used in the code:

A close-up of a computer screen

AI-generated content may be incorrect.Where:

time: Used to measure the encrypting and decrypting time

[matplotlib.pyplot](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Used to draw performance graph.

[Crypto.PublicKey.RSA](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Used to generate and manage RSA keys

[Crypto.Cipher.PKCS1\_OAEP](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Used to perform RSA encryption and decryption in PKCS1\_OAEP mode.

Crypto.Random.get\_random\_bytes: Used to generate a random message of a specific length.

A close-up of a code

AI-generated content may be incorrect.Define **generate\_keys()** function to generate private and public keys

**Purpose**: Generate an RSA key pair (public key and private key).

**Note that:**

* [RSA.generate(2048)](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): generate an RSA key pair with a length of 2048 bit. It must be at least 1024, but **2048 is recommended.** Note that **RSA.generate(bits)** will generate a different key each time it is run.
* [key.export\_key()](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Export private key as a byte string.
* [key.publickey().export\_key()](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Export public key as a byte string.
* Return: A tuple contains private and public keys.

A computer code on a white background

AI-generated content may be incorrect.Define **encrypt\_message(public\_key, message)** function to encrypt the message:

**Purpose:** Encrypt the message with RSA pubic key.

**Note that:**

* [RSA.import\_key(public\_key)](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Import public key from byte string.
* [PKCS1\_OAEP.new(rsa\_key)](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Create new encrypted RSA object with PKCS1\_OAEP mode.
* chunk\_size = rsa\_key.size\_in\_bytes() - 42: Calculate the maximum size of each message chunks (decrease 42 bytes for padding).

Then, perform a loop:

* Split message into small chunks which is smaller than chunk\_size.
* Encrypt each chunk and concatenate the encrypted chunks.
* Return: Encrypted message (ciphertext).

A screen shot of a computer code

AI-generated content may be incorrect.Define **decrypt\_message(private\_key, ciphertext)** function to decrypt the message:

**Purpose:** Decrypt encrypted message by using RSA private key.

**Note that:**

* [RSA.import\_key(private\_key)](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Import public key from byte string.
* [PKCS1\_OAEP.new(rsa\_key)](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Create new encrypted RSA object with PKCS1\_OAEP mode.
* chunk\_size = rsa\_key.size\_in\_bytes(): Calculate the size of each ciphertext chunk.

Then, perform a loop:

* Split ciphertext (encrypted message) into small chunks whose size is equal to chunk\_size.
* Decrypt each chunk and concatenate the decrypted chunks.
* Return: Original message (plaintext).

Define **measure\_performance()** function to measure the performance, data statistics and draw graph:

**Purpose:** Measure encrypt and decrypt tine of messages with different lengths.

**Note that:**

* [message\_lengths](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): List of messages lengths (from 0 to 5120 bytes, with step equals 256 bytes).
* [get\_random\_bytes(length)](vscode-file://vscode-app/c:/Users/LG/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Generate a random message of length **length (byte)**.

**Measure time:**

* Encrypt: Record the start and end times when encrypting the message.
* Decrypt: Record the start and end times when decrypting the ciphertext.
* Output: Print encrypting and decrypting time of each message with different lengths.

**Draw graph:**

X axis: Lengths of messages (bytes).

Y axis: Encrypting and decrypting time (second).

## Discussion

## Recommend

# REFERENCES

Eijs, H. (2014, June 24). *PyCryptodome 3.22.0 documentation*. Retrieved from Read the Docs: https://pycryptodome.readthedocs.io/en/latest/

Johnson, J. (2001, September 26). *Logical connective*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Logical\_connective

Szopen. (2002, January 10). *Reverse Polish notation*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Reverse\_Polish\_notation

WojPob. (2001, February 21). *RSA cryptosystem*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/RSA\_cryptosystem