Sample Edition

ITPEC Fundamentals of Information Technology Engineer Examination (FE)

– English Edition October 2025 Subject B Version

Author: Takashi Narita

Based on past **FE Examination** questions published by ITPEC.

Disclaimer

This book contains past exam questions from the Fundamental Information Technology Engineer Examination (FE), administered by the Information Technology Professionals Examination Council (ITPEC), specifically from the October 2025 Examination. The questions are reproduced with permission for educational purposes, and the copyright of the original questions remains with ITPEC.

All explanations, translations, and analyses are original work by the author, Takashi Narita.

Preface

The **Fundamental Information Technology Engineer Examination (FE)** is a core examination that demonstrates your ability to apply fundamental knowledge of information technology, programming, algorithms, and system design. Originally developed in Japan, it is now recognized internationally through the ITPEC (Information Technology Professionals Examination Council) mutual recognition framework. Member countries include the Philippines, Thailand, Vietnam, Myanmar, Mongolia, and Bangladesh.

In many of these countries, the FE serves as a gateway for those aiming to work in Japan or for Japanese companies abroad. Where Japanese-owned IT firms operate, holding this certification can also provide a strong advantage in hiring and career advancement, as it reflects both technical competence and familiarity with Japanese corporate culture and IT standards.

This book is designed for learners who wish to prepare for the **FE Examination in English**. It offers clear explanations, answer analyses, and study tips to help you build genuine understanding rather than relying on rote memorization. Whether you are a student, a working professional, or someone seeking to advance your IT career, this book will support efficient, practical, and motivating study.

Important Notice

The ITPEC Fundamentals of Information Technology Engineer Examination (FE) consists of two parts: Subject A and Subject B.

Each subject has different characteristics and types of questions.

This book covers only Subject B.

➞ For Subject A, please purchase the separate volume.

➞ A combined edition (Subjects A & B) is also available.

About the Author

Takashi Narita is an IT instructor with more than 20 years of industry experience in system design, software development, programming, and database management. Now based in the Philippines, he teaches programming, system development, and system design remotely to students in Japan.

He began creating this English edition of **Fundamental Information Technology Engineer Examination (FE)** practice questions to support IT human resource development in the Philippines. Over time, he recognized that it could also serve as a valuable resource for learners across Asia seeking to succeed in Japanese-owned companies or in Japan itself.

Through this book, he aims to help readers build the skills and confidence needed to contribute meaningfully in Japanese and Japan-affiliated workplaces. He is passionate about making complex IT concepts accessible to learners of all backgrounds and believes that studying in English opens doors to international opportunities.

How to Use This Book

The fastest way to pass the **Fundamental Information Technology Engineer Examination (FE)** is to practice past questions repeatedly and understand the logic behind each answer.

This book provides past FE questions in English, along with clear explanations and answer analyses. Always review the explanations for any questions you miss, make sure you understand the reasoning, and then try again.

Recommended study flow:

1. Solve one set of questions under timed conditions, just like the actual exam.
2. Check which answers were incorrect and study the explanations carefully.
3. Reattempt the same set a few days later until you can achieve a high score with confidence.
4. Move on to the next set and repeat the process.

Study Tips

• **Do more than one set** – One set of past questions is not enough for thorough preparation. Aim for at least three sets to build both knowledge and confidence.

• **Consistency is key** – Even 10 minutes a day makes a difference. Use your phone, PC, or tablet to open this book and solve just one question. From my own experience earning multiple IT certifications, daily habit is the single most important factor in success. For me, solving one question before bed soon grew naturally to two or three without feeling forced. Short, consistent study sessions lead to long-term retention.

• **Set a study schedule** – Work backward from your exam date, create a plan, and track your progress.

• **Get used to English IT terms** — many are 3–4-letter abbreviations (e.g., CPU, LAN, DNS) and can feel intimidating at first, but with repeated exposure they'll become second nature.

Exam Overview

• **Format: Two parts** — **Subject A** (formerly “Morning”) and **Subject B** (formerly “Afternoon”).

• **Exam Areas (Subject A):** 60 multiple-choice questions in 90 minutes.

Subject A consists of **60 questions** drawn from the three domains below.

1) Technology Domain

IT fundamentals, hardware, software, databases, networks, security, etc.

2) Management Domain

project management, service management, system audit, etc.

3) Strategy Domain

management strategy, business law, system strategy, corporate activities, etc.

You should aim to spend about 1.5 minutes (90 seconds) per question.

• **Exam Areas (Subject B):** 20 multiple-choice questions in 100 minutes.

Algorithms & Programming (pseudo-code); Information Security.

(**20 questions** total: 16 Algorithms & Programming, 4 Information Security)

You should aim to spend about 5 minutes per question.

• **Passing Criteria (FE):**

You must pass both Subject A and Subject B. The passing mark for each subject is 60 out of 100.

Links

The test mode (CBT or paper), schedule, application method, and venue vary by country. Details are subject to change; always check your country's official website for the latest information.

Official Information Links:

- ITPEC Official Site: https://itpec.org/
- Philippines (DICT): https://dict.gov.ph/itpec/
- Thailand (NECTEC): https://www.nectec.or.th/itpec/
- Vietnam (VITC): https://www.vitc.vn/
- Myanmar (ITPEC Myanmar): https://www.myanmaritpec.org/
- Bangladesh (BCC): https://www.bcc.gov.bd/
- Mongolia (MITC): https://www.mitc.gov.mn/

Author's Note: A quick word on Subject B

Subject B tests **programming & algorithms**.

Unlike Subject A—where past-like items can be solved by "recognition"—you'll **rarely** see the exact same algorithm. What matters is **understanding and application**.

When it feels hard — do this

1. Peek if needed, fill the blanks, then **trace the finished code** yourself.
2. Use **tiny inputs** and a quick table (track i, j, sum, max, …).
3. Check **stopping conditions & boundaries** (loop end, indices, empty/one/many).
4. Learn the **shape** of formulas (e.g., relative error, Hamming distance)—deep math not required.

With these habits, fill-in questions get much easier. Unfamiliar names won't stop you if you can **trace correctly**.

Bottom line: Subject B is often the hurdle. **Master basics, repeat patterns, move one steady step at a time.**

Note: Styles and coverage can change—review the latest syllabus and past papers.

*ITPEC provides an official **Pseudo-code Specification** in the downloadable past exams. For authoritative syntax/semantics, refer to those documents.*

Exams are always a race against time. If you glance at a question and feel a strong sense of dread, move on to another one. The best strategy is to start with the questions that make you think, "I can do this!"

B-Q1

From the answer group below, select the correct combination of answers to be inserted into ____A____ and ____B____ in the program. Here, the array index starts at 1.

The function sum receives an integer array array with at least two elements and two positive integers k and m (k < m ≤ number of elements in array). It calculates the sum of even elements of the array array whose indices are from k to m.

[Program]

```
integer: sum(integer []: array, integer: k, integer: m)
  integer: s ← 0
  integer: i ← k
  while (i ≤ m)
    if (    A    )
      s ← s + array[i]
    endif
    B
  endwhile
  return s
```

Answer group

	A	B
a)	array[i] mod 2 = 0	i ← i + 1
b)	array[i] mod 2 = 0	i ← i - 1
c)	array[i] mod 2 ≠ 0	i ← i + 1
d)	array[i] mod 2 ≠ 0	i ← i - 1

(Source:2025A,FE,Subject-B,Q1)

Answer: a

Note to Learners:

This question has a low difficulty level. Consider it a "lucky question" and make sure to secure these points for your score!

Points to Focus:

The problem is straightforward if you focus on these logic triggers:

1. **Target:** "Sum of even elements" → You need a condition to check for even numbers.
2. **Range:** "From ***k*** to ***m***" → The loop starts at ***k*** and must move toward ***m***.
3. **Loop Control:** Any `while` loop processing an array needs a way to move to the next index, or it will run forever!

Explanation

The function calculates the sum of even numbers within a specific range of an array.

- **Blank A (Even Number Check):** To determine if a number is even, we check if the remainder is 0 when divided by 2. In pseudo-code, the `mod` operator is used for this.
  Therefore, `array[i] mod 2 = 0` is the correct condition.
- **Blank B (Index Update):** The loop begins with ***i*** set to ***k*** and continues as long as $i \leq m$. To ensure the program checks every element in the range and eventually finishes, the index `i` must be increased by 1 in each step. Thus, `i ← i + 1` is required.

Column: Mastering the "mod" Operator

The mod (modulo) operator is a staple in programming logic, used to find the remainder of a division. It is most frequently used for parity checks:

- **Even:** `n mod 2 = 0`
- **Odd:** `n mod 2 ≠ 0`

Message from the Author:

When you read the explanation, you probably think, "Ah, I see." But when you actually encounter a problem like this, there won't be any explanation provided. Can you fill in the blanks on your own? One effective method is to **rule out the wrong answers.** Look at the answer choices.

	A	B
a)	`array[i] mod 2 = 0`	`i ← i + 1`
b)	`array[i] mod 2 = 0`	`i ← i - 1`
c)	`array[i] mod 2 ≠ 0`	`i ← i + 1`
d)	`array[i] mod 2 ≠ 0`	`i ← i - 1`

For the first blank, A, there are only two possibilities, right?

"array[i] mod 2 = 0" and **"array[i] mod 2 ≠ 0."**

What happens when this IF statement evaluates to True?

That's right — it adds a value to the variable **S**.

What was this program supposed to do?

It was supposed to add up the even numbers, wasn't it?

Then blank A just needs to check whether the number is even.

So the answer is obvious. There is only one correct choice:

"array[i] mod 2 = 0," right?

Now, let's move on to blank B.

The variable is either incremented by +1 or decremented by −1.

But look at the termination condition of this WHILE loop: **(i ≤ m)**.

If you subtract 1 from i, you'll end up in an infinite loop!

So there's only one possible answer — it must be +1.

That's how you can determine the correct answer.

Fill in the blanks with the answer you come up with, and then trace the program yourself.

If you practice this approach, you'll be able to handle Subject B much more confidently.

Advice for Those Who Struggle with the Programming Questions in Subject B

I am a lecturer at an IT vocational school in Japan and have been teaching Information Technology Passport (FE) exam courses for many years. Many students find algorithm problems (Subject B) challenging because the skills required are fundamentally different from those in Subject A. The biggest question candidates face is: "How can I build my algorithm skills?".

Let me teach you the absolute basics. I have filled in the blanks of the program with the correct answers, so it will work perfectly. Now, I want you to **trace** it. This is the exact method I use in my actual classes for students who struggle with algorithms. Don't say, "There's no point if the answers are already there!". This is an exercise in tracing a program to build your **algorithm muscles!**

Let's trace this logic using specific values:

- **Setup:** Array {1, 2, 3, 4, 5}, ***k*** = 1, ***m*** = 5.
- **Goal:** Sum of even numbers (2 + 4) = **6**.

```
1  integer []: array ← {1, 2, 3, 4, 5}
2  integer: k ← 1
3  integer: m ← 5
4  integer: s ← 0
5  integer: i ← k
6  while (i ≤ m)
7    if (array[i] mod 2 = 0)
8      s ← s + array[i]
9    endif
10   i ← i + 1
11 endwhile
12 return s
```

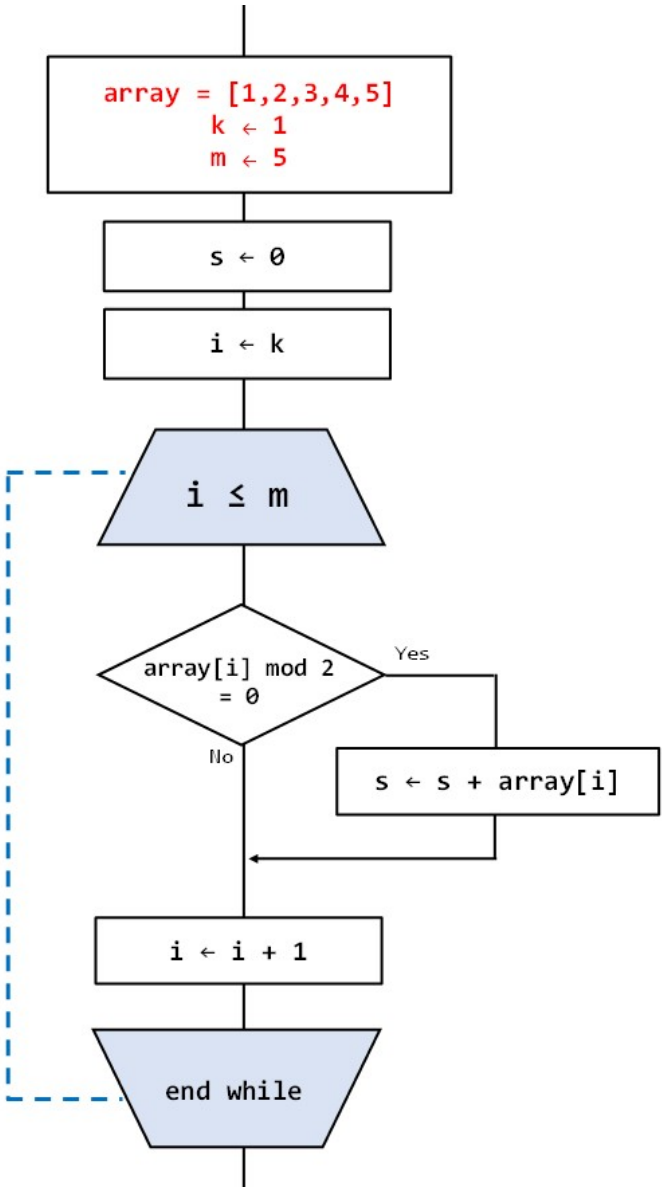
Step-by-Step Execution:

To build your "algorithm muscles," I give you a advice. **"Don't ask why."** Just trace the code as it is written.

1. **Line 4:** s ← 0 (Initialize the sum).
2. **Line 5:** ***i*** ← ***k*** (***i*** becomes 1).
3. **Line 6:** while (***i*** ≤ ***m***) (**1≤5** is True, so enter the loop).
4. **Line 7:** if (array[***i***] mod 2 = 0)
    - array[1] is **1**. 1 mod 2 = 1 (Odd). The condition fails.
5. **Line 10:** ***i*** ← ***i*** + 1 (***i*** becomes 2).
6. **Line 6:** while (***i*** ≤ ***m***) (**2≤5** is True, so enter the loop).
7. **Line 7:** array[2] is **2**. 2 mod 2 = 0 (Even). The condition is **True**!.
8. **Line 8:** s ← s + array[2] (s becomes 0 + 2 = 2).
9. **Line 10:** i becomes 3.

...This continues until ***i*** becomes 6, making the condition 6**≤5** False. The loop ends, and we return s.

I have drawn a flowchart for you. I believe it makes the flow of the process much easier to visualize. Now, should you trace the source code? Or should you trace the flowchart? In the beginning, either way is fine. Please start with whichever one you prefer!



As I said at the beginning: **"Don't ask why."** Just execute the commands like a CPU. A CPU never asks, "Why are you telling me to do this?" (laughs).

But now that you've finished the trace, doesn't everything become clear?

- "The **`while`** condition is checking if we've reached the end of the array!"
- "Adding 1 to **`i`** is how we move to the next item!"
- "We started **`s`** at 0 because it needs to be empty before we start adding!"
- `. . .`

If you can see these patterns now, you've passed the first step. If not, keep tracing until that "Aha!" moment hits you.

B-Q2

From the answer group below, select the correct combination of answers to be inserted into ___A___ and ___B___ in the program. Here, the array index starts at 1.

In statistics, the mode is the value that occurs most frequently in a dataset. For example, the mode of the integer array {2, 1, 1, 9, 6, 6, 2, 5, 6} is 6, as it occurs most often. The function findMode receives an integer array arr as a dataset and returns the mode for it.

[Program]

```
integer: findMode(integer []: arr)
  integer: n ← the number of elements in arr
  integer: m ← arr[1]   /* Current mode value */
  integer: m_c ← 1      /* Frequency count of mode */
  integer: c, i, j
  for (increase i from 1 to n - 1 by 1)
    c ← 1
    for (increase j from i + 1 to n by 1)
      if (    A    )
        c ← c + 1
      endif
    endfor
    if (    B    )
      m_c ← c
      m ← arr[i]
    endif
  endfor
  return m
```

Answer group

	A	B
a)	arr[i] = arr[j]	m < arr[i]
b)	arr[i] = arr[j]	m ≠ arr[i]
c)	arr[i] = arr[j]	m_c < c
d)	arr[i] = arr[j]	m_c > c
e)	m = arr[j]	m < arr[i]
f)	m = arr[j]	m ≠ arr[i]
g)	m = arr[j]	m_c < c
h)	m = arr[j]	m_c > c

(Source:2025A,FE,Subject-B,Q2)

Answer: c

Note to Learners: This question is a step up from the first one, but don't worry! It’s all about counting and comparing. Once you understand how the nested loops work, you'll see this is a very logical and high-scoring question to master!

Points to Focus: To solve this like a pro, focus on these two "counting" logic triggers:

1. **The Matching Rule:** To count how many times a number appears, you have to compare the target number with every other number in the list.
2. **The High-Score Rule:** To find the "winner" (the mode), you compare the current count to the highest count you've found so far. If the new count is higher, you have a new winner!

Explanation

- **Blank A (Counting Occurrences):** The outer loop (index `i`) picks a target value, and the inner loop (index `j`) scans the rest of the array to count how many times that target value appears. To increment the frequency counter `c`, we must check if the value at the current position `j` matches our target value at position `i`. Therefore, the correct condition is `arr[i] = arr[j]`.
- **Blank B (Updating the Mode):** After the inner loop finishes, `c` holds the frequency of the current value `arr[i]`. We compare this to `m_c`, which stores the highest frequency found so far (the "current mode's count"). If the new count `c` is greater than the record `m_c`, it means we have found a new mode. Thus, the condition to update the mode is `m_c < c`.

Message from the Author:

You may be one of the few who can fully understand this problem just by reading the explanation. As I've mentioned before, if you don't understand it, insert the correct answer choice and trace the program step by step yourself. Then make sure you understand why it works correctly.

This kind of practice is essential for improving your accuracy in Subject B. In particular, this problem deals with a **nested loop**—something that almost everyone struggles with at first, and which always appears on the FE exam.

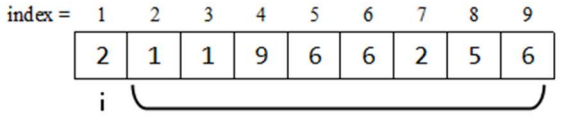
```
1  integer: findMode(integer []: arr)
2    integer: n ← the number of elements in arr
3    integer: m ← arr[1]   /* Current mode value */
4    integer: m_c ← 1      /* Frequency count of mode */
5    integer: c, i, j
6    for (increase i from 1 to n - 1 by 1)
7      c ← 1
8      for (increase j from i + 1 to n by 1)
9        if ( arr[i] = arr[j] )
10         c ← c + 1
11       endif
12     endfor
13     if ( m_c < c )
14       m_c ← c
15       m ← arr[i]
16     endif
17   endfor
18   return m
```

The loop on line 6 uses the variable **i.** The condition runs from **1** to **n−1** — in other words, from the beginning up to one element before the end.

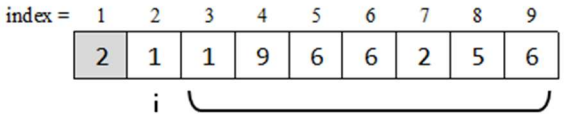
The loop condition on line 8 uses the variable **j**. The condition runs from **i+1** to **n** — that is, from the position immediately after the loop variable **i** (in line 6) to the last element.

Understanding this relationship is the most important point!

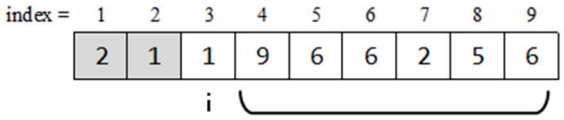
Let's trace it from the beginning.



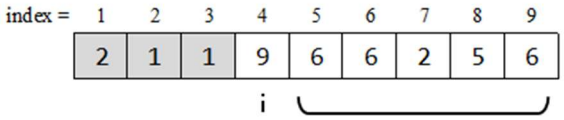
First, in the outer loop, the variable **i = 1**.
The inner loop (variable **j**) runs from 2 to the end.
As a result, the element at index = 1, which is "2," is checked against the range covered by the inner loop (index = 2 to 9). It counts how many times "2" appears in that range, and the result is stored in the variable **c**.
Since the initial value of **c** is 1, it becomes 2.
Now, look at the if statement on line 13.
Because **m_c = 1** (initial value) and **c = 2**, the condition on line 13 is True. Therefore, **m_c = 2**, and **m=2.**
This means that "2" appears twice.



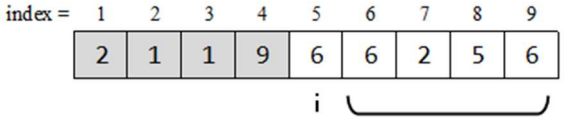
Next.
The value "1" at index=2 is found once between index= 3 and 9. So **c = 2**. Line 13 is False.



Next.
The value "1" at index=3 is not found between index= 4 and 9. So **c = 1**. Line 13 is False.



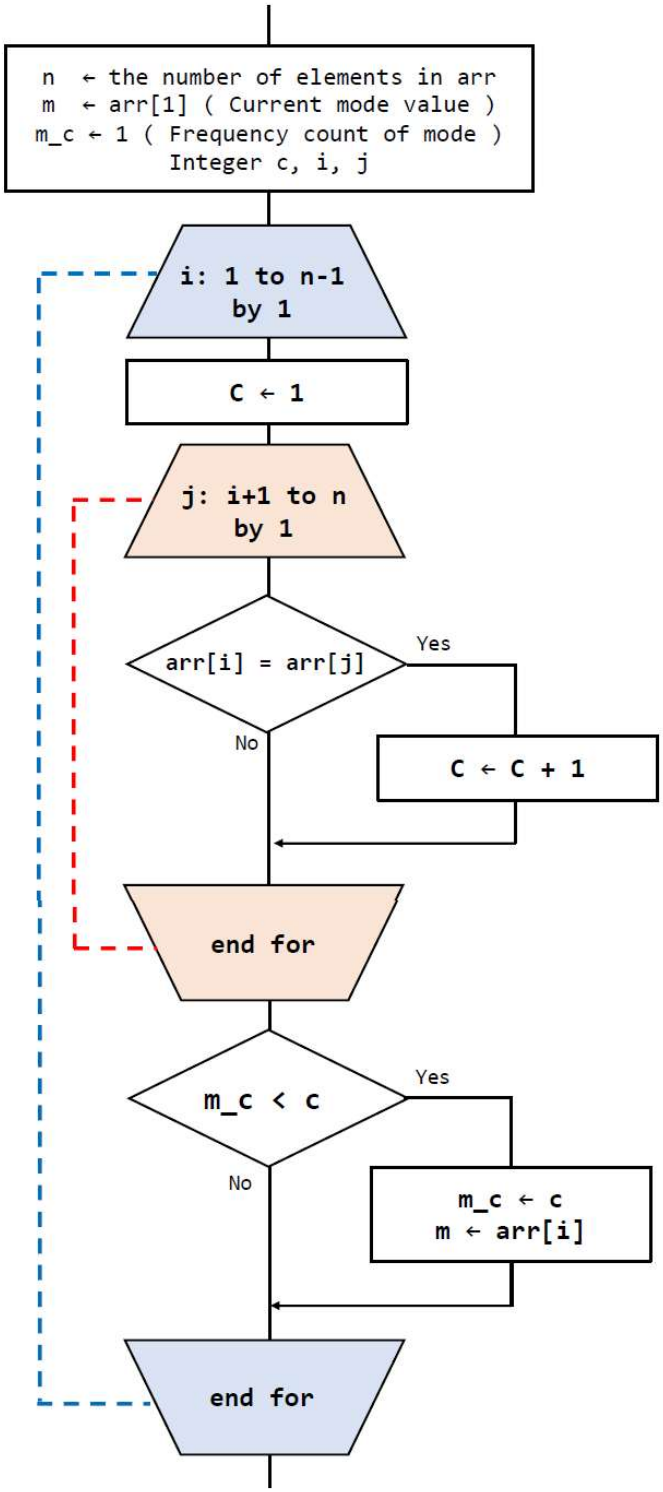
Next.
The value "9" at index=4 is not found between index= 5 and 9. So **c = 1**. Line 13 is False.



Next.
The value "6" at index=5 is found twice between index = 5 and 9. So **c = 3**. Line 13 is True.
m_c=3, and **m=6**.
This means that "6" appears 3times.

The rest is omitted.

In this way, based on how the outer loop (variable **i**) and the inner loop (variable **j**) move, can you see how the process proceeds? And can you now see that the purpose "**The value that occurs most frequently**" of this program is being achieved?


n ← the number of elements in arr
m ← arr[1] (Current mode value)
m_c ← 1 (Frequency count of mode)
Integer c, i, j
i: 1 to n-1 by 1
C ← 1
j: i+1 to n by 1
arr[i] = arr[j]
Yes
No
C ← C + 1
end for
m_c < c
Yes
No
m_c ← c
m ← arr[i]
end for

This Book is sample edition.

Notes and Disclaimers

Source Note:

The questions in this book are based on the past questions from the October 2025 FE (Fundamentals of Engineering) Examination published on the official ITPEC website. The author translated them into English and added supplementary explanations.

Source: ITPEC – FE Examination Past Questions (https://itpec.org/)

Relationship with ITPEC:

This book is an independent publication. It is created in accordance with the usage policy for past exam questions published by ITPEC, but it is **not** officially affiliated with, authorized, or endorsed by ITPEC.

General Disclaimer:

The content of this book is based on information available at the time of writing. The examination format, scope, or content may change. Please refer to the official ITPEC website for the most up-to-date **information.**

License and Usage:

This book is intended for personal study purposes. Redistribution or commercial use of the explanations and translations by the author without permission is prohibited.

Feedback Welcome:

If you find any errors or have suggestions for improvement, please feel free to contact the author. Your input will help improve future editions of this book.

Copyright & Source Information