1.5. RECURSION VS ITERATION

Recursion and Iteration both repeatedly execute the set of instructions. **Recursion** occurs when a statement in a function calls itself repeatedly. The iteration occurs when a loop repeatedly executes until the controlling condition becomes false. The basic difference between recursion and iteration is that recursion is a process always applied to a function and iteration is applied to the set of instructions which we want to be executed repeatedly.

Read through this article to find out more about Recursion and Iteration and how they are different from each other.

What is Recursion:

Recursion is defined as a process in which a function calls itself repeatedly. Recursion uses selection structure. If the recursion step does not reduce the problem in a manner that converges on some condition, called **base condition**, then an **infinite recursion** occurs. An infinite recursion can crash the system. Recursion terminates when a base case is recognized.

Because of the overhead of maintaining the stack, the recursion process is usually slower than iteration. Also, recursion uses more memory than iteration. However, it makes the code smaller, thus it is an amazing technique that makes it easier to read and write the code.

What is Iteration:

Iteration is defined as the repetition of computational or mathematical procedure that continues until the controlling condition becomes false. It uses repetition structure. If the loop condition test never becomes false, then an infinite loop occurs with iteration. This infinite looping uses **CPU** cycles repeatedly. An iteration terminates when the loop condition fails. Iteration consumes less memory, but makes the code longer which is difficult to read and write.

Difference between Recursion and Iteration

The following table highlights all the important differences between recursion and iteration ?

Recursion	Iteration

Recursion uses the selection structure.	Iteration uses the repetition structure.
Infinite recursion occurs if the step in recursion doesn't reduce the problem to a smaller problem. It also becomes infinite recursion if it doesn't convert on a specific condition. This specific condition is known as the base case.	An infinite loop occurs when the condition in the loop doesn't become False ever.
The system crashes when infinite recursion is encountered.	Iteration uses the CPU cycles again and again when an infinite loop occurs.
Recursion terminates when the base case is met.	Iteration terminates when the condition in the loop fails.
Recursion is slower than iteration since it has the overhead of maintaining and updating the stack.	Iteration is quick in comparison to recursion. It doesn't utilize the stack.
Recursion uses more memory in comparison to iteration.	Iteration uses less memory in comparison to recursion.
Recursion reduces the size of the code.	Iteration increases the size of the code.

Difference between Iteration and Recursion

Property	Recursion	Iteration
Definition	Function calls itself.	A set of instructions repeatedly executed.

Property	Recursion	Iteration
Application	Certain problems can be solved quite easily using recursion like Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.	In general iterative solutions are preferred over recursive solutions as there is no extra overhead required for recursion.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used in academics to teach foundations and logic. Also works as a foundation for Dynamic Programming and Divide and Conquer algorithms.	Preferred in general.
Code Size	Can be smaller code size for inherently recursive problems.	Can be larger for naturally recursive problems.
Time Complexity	In general time complexity is higher or same. The time complexity especially become higher for the problems that can be optimized using dynamic programming	Lower or same.
Space Complexity	Auxiliary space is either higher or same. Can be same in same cases where we need an explicit stack to simulate recursion like tree traversals, merge sort, etc.	Generally lower

Property	Recursion	Iteration
Overhead	Possesses overhead of repeated function calls.	No overhead as there are no function calls in iteration.

REAL-WORLD USE CASES: BALANCED PARENTHESES

WHAT ARE BALANCED PARENTHESES?

Balanced parentheses mean that every opening bracket has a corresponding closing bracket in the correct order.

Examples

Balanced

()

([ ])

{()}

((A+B)*C)

Not Balanced:

(()

([ )]

{(

Balanced parentheses are usually checked using a **Stack** data structure.

Why is Balanced Parentheses Checking Important?

Many computer systems need to verify that symbols such as:

()

{}

[]

<>

are correctly matched before processing data.

Incorrect matching can lead to:

- Compilation errors
- Program crashes
- Invalid expressions
- Incorrect document structures

1. Compiler Design and Syntax Checking

One of the most important applications of balanced parentheses is in **compilers**.

Example

```
if(a > b)
{
   printf("Hello");
}
```

The compiler checks whether:

```
{
}
```

are balanced.

Incorrect Code

```
if(a > b)
{
```

```
printf("Hello");
```

Missing:

}

Compiler Error:

Expected closing brace

2. Expression Evaluation

Mathematical expressions must contain balanced parentheses.

Example

Valid:

((5+3)*2)

Invalid:

((5+3)*2

Before evaluating an expression, the system checks bracket balance.

Applications

- Scientific calculators
- Mathematical software
- Spreadsheet systems

Examples:

- Microsoft Excel formulas
- Engineering calculators

3. Integrated Development Environments (IDEs)

Modern IDEs automatically check matching brackets while coding.

Example

When typing:

public class Test {

The IDE highlights:

{

and expects:

}

Popular IDEs

- Visual Studio Code
- Eclipse
- IntelliJ IDEA

Benefit

- Prevents coding mistakes
- Improves productivity

4. HTML and XML Document Validation

HTML and XML use opening and closing tags similar to parentheses.

Example

Valid HTML

```
<html>
  <body>
    <h1>Hello</h1>
  </body>
</html>
```

Invalid HTML

```
<html>
  <body>
    <h1>Hello
  </body>
</html>
```

Missing:

</h1>

Applications

- Web browsers
- Website validation tools
- XML parsers

Benefit

Ensures proper document structure.

5. Database Query Processing

SQL queries often contain nested parentheses.

Example

```
SELECT *
FROM Employee
WHERE (Salary > 50000 AND
    (Department='IT' OR Department='HR'));
```

Database systems verify bracket matching before executing queries.

Applications

- Database Management Systems (DBMS)
- Query Optimizers

Benefit

Prevents query execution errors.

6. Programming Language Parsing

Programming languages contain many nested structures.

Example

```
while(a > 0)
{
   if(a > 5)
   {
      System.out.println(a);
   }
}
```

Parser checks:

()
{}
[]

Applications

- Language translators
- Interpreters
- Compilers

Benefit

Creates correct syntax trees.

7. Text Editors and Code Editors

Editors automatically identify matching brackets.

Example

Typing:

(

The editor highlights:

)

Applications

- Code editors
- Document editors

Features

- Auto-completion
- Error highlighting
- Code formatting

8. Expression Conversion

Balanced parentheses are necessary for converting:

Types

Infix

A + B * C

Postfix

ABC*+

Prefix

+A*BC

Before conversion, parentheses balance must be verified.

Applications

- Compiler design
- Calculator software

9. JSON and Data Validation

JSON files contain nested structures.

Example

Valid JSON

```
{
  "name":                                    "John",
  "age":                                        25
}
```

Invalid JSON

```
{
  "name":                                    "John",
  "age": 25

Missing:

}
```

Applications

- APIs
- Web services
- Configuration files

Benefit

Ensures correct data exchange.

10. Artificial Intelligence and Natural Language Processing (NLP)

AI systems process structured text containing brackets.

Example

Mathematical expression:

((A+B)*(C-D))

AI models first verify structural correctness.

Applications

- Chatbots
- Formula parsers
- Symbolic computation systems

How Stack Solves Balanced Parentheses

Algorithm

1. Read expression from left to right.
2. Push opening brackets onto stack.
3. For every closing bracket:
    - Pop from stack.
    - Check matching pair.
4. If stack becomes empty at the end → Balanced.

Example

Expression:

{[()]}

Stack Operations

Symbol	Operation	Stack
{	Push	{
[	Push	{[

Symbol	Operation	Stack
(	Push	{[(
)	Pop	{[
]	Pop	{
}	Pop	Empty

Result:

Balanced

Complexity Analysis

Let **n** be the number of symbols.

Time Complexity
O(n)

Each symbol is processed once.

Space Complexity
O(n)

In the worst case, all opening brackets are stored in the stack.

Advantages

- Detects syntax errors quickly.
- Essential for compilers and parsers.
- Ensures data integrity.
- Helps maintain correct program structure.
- Improves software reliability.

UNDO-REDO SYSTEMS