Application of Stack Data Structure in Application Development

Sarah Amaylia[1], Viktoria Angelita Setiabudi[2], Renza Alvianino[3], Rahmat Nugroho Saputra[4], Helena Kusuma Wardhani [5],Aziz Suroni[5].
State University of Surabaya, Indonesia
{24111814017, 24111814066, 2411814011, 2411814042, 2411814020}@mhs.unesa.ac.id [1][2][3][4][5], azissuroni@unesa.ac.id[6]

Abstract— The rapid development of digital technology has driven the need for efficient, modular, and maintainable applications. In this context, the stack data structure plays a crucial role in supporting the development of responsive and structured applications. This article discusses the application of the stack data structure in various aspects of application development, both conceptually and technically. The method used is a literature review, analyzing nine scientific journals as the basis for the study.

The results indicate that the stack is not only theoretically relevant as a Last-In First-Out (LIFO) structure but also proven effective in implementing various application features, such as user navigation, undo-redo systems, mathematical expression processing, and graph traversal. Stacks can be implemented using arrays or linked lists, depending on the requirements for flexibility and memory efficiency.

Considering its ease of implementation and wide range of real-world applications, the stack data structure becomes an important component for developers to master. Proficiency in stack concepts and practices directly contributes to improving the quality of applications built.

Keywords— Stack, Data Structure, Python, Application Development.

I. INTRODUCTION

The rapid development of digital technology in recent years has brought about significant changes in how humans interact with information and computing systems. This transformation has not only impacted daily life but also revolutionized the approach to software development. Continuous digital innovations demand software that is not only efficient but also scalable, maintainable, and adaptable to dynamic user needs. In this context, the selection and implementation of appropriate data structures become a crucial component that cannot be overlooked.

A data structure defines a method of storing, organizing, and arranging data within a computer storage medium so that the data or information can be used efficiently. In programming techniques, a data structure refers to the layout of data containing data columns, whether visible to the user or not[1][2]. It is stated that building the right data structure will enhance algorithmic capabilities, while developing the right algorithm will reduce the complexity of the algorithm itself[3]. Without a good understanding of data structures, developers will find it difficult to build optimal systems, especially when faced with large-scale challenges or high system complexity. One of the fundamental yet highly influential data structures is the stack

A stack is a linear data structure that operates on the principle of Last In, First Out (LIFO)[4]. This means the last element added to the stack is the first one to be removed[5]. While this principle seems simple, it's very powerful and has many practical applications in programming. A stack provides two main operations: push, which adds an element to the top of the stack, and pop, which removes an element from the top of the stack[6]. Despite having only these two core operations, stacks are a crucial component in various algorithms and computer system mechanisms.

In the real world, the stack data structure is a crucial part of many systems and applications, even though its use is often hidden from the end-user. For instance, in the function call systems of modern programming languages, whenever a function is called, essential information like function parameters, local variables, and return addresses are saved onto a stack. This mechanism allows functions to be called in a nested or recursive manner while maintaining data consistency and program flow. In this context, the stack functions as a call stack, which is vital for managing program execution flow.

Beyond the technical backend, stacks are also applied in various user interface features that are very familiar to everyday users.The most common example is the undo and redo features in word processors or graphic editors[7]. Every user action is stored in a

stack, allowing it to be undone or redone in the correct sequence[8][9]. Stacks are also used in web page navigation, where browsers store the history of visited pages in two stacks: one for the "back" stack and another for the "forward" stack[10]. When a user presses the "back" button, the browser retrieves the last page from the back stack and adds it to the forward stack. Stacks can even be used in music queuing[11].

In mobile app development, modern frameworks like React Native also rely on the stack concept to manage screen navigation. Whenever a user moves from one screen to another, the previous screen is saved onto a stack. This allows users to easily return to previous screens. This concept is known as a navigation stack[12].

Despite its importance, many software developers still don't fully optimize their use of the stack data structure. This can stem from various factors, such as a limited understanding of data structure theory, insufficient practical implementation experience in real-world projects, or the dominance of frameworks that explicitly hide data structure usage. As a result, many developers tend to rely heavily on automatic framework features without understanding the underlying logical processes. Ultimately, this can hinder their ability to solve complex problems or perform system optimizations.

Given the crucial role of stacks in various aspects of software development, a deeper exploration of their working principles, implementation, and practical applications is necessary. This article aims to provide a conceptual and technical understanding of stacks, from their definition and operations to case studies of their implementation in real-world applications. By doing so, we hope developers, students, and academics will revisit the importance of mastering this data structure as a foundation for building more efficient, reliable, and sustainable systems.

Data structures aren't just theoretical concepts in computer science curricula; they are the tangible foundation of logic and efficiency in modern software development. The stack, as one of the most fundamental data structures, demonstrates how conceptual simplicity can yield immense power in practical implementation. Through deep understanding and proper application, the stack data structure can become a strategic tool for designing robust and adaptive systems amidst evolving technological challenges.

II. RESEARCH METHODOLOGY

This research was conducted through several stages of experimental testing of stacks in programming languages related to system development. The research stages for this method are as follows:

A. Research Variables

Table 1 **Research Variables**

No	Variable Name	Operational Definition	Measurement Scale	Data Collection Method
1.	Data Size	Number of elements in stack operations	Ratio	Programmatic counter
2.	Operation Type	Type of stack operation (push/pop)	Nominal	Experimental control

B. Literature Study

The stack data structure has been the subject of intensive research in computer science due to its simple yet powerful nature. The following is a literature review related to stack implementation in the context of application development:

1) Basic Concepts of Stack

A stack is a linear data structure that follows the LIFO (Last In, First Out) principle, where the last element inserted will be the first to be removed. LIFO (Last In, First Out) means the last element pushed will be the first to be popped. Example: A stack of plates—the top plate is taken first.

A stack has several fundamental operations:

1. Push (Insert Data)

Function: Adds a new element to the top of the stack.

Error Condition: If the stack is full (stack overflow), the push operation fails. Example: Python implementation: stack.push(5) # Stack: [5] stack.push(10) # Stack: [5, 10]

2. Pop (Remove Data)

Function: Removes an element from the top of the stack and returns its value.

Error Condition: If the stack is empty (stack underflow), the pop operation fails.

Example:

stack.pop() # Returns 10, Stack becomes [5]

3. Peek / Top (View Top Data)

Function: Returns the value of the top element without removing it.

Error Condition: If the stack is empty, the peek operation returns null or an error.

Example:

stack.peek() # Returns 5 (stack remains [5])

4. isEmpty (Check If Stacks Is Empty) Function: Checks if the stack is empty.

Return:

True if the stack is empty.

False if the stack contains elements.

Example:

stack.is_empty() # False (because it still has [5])

5. isFull (Check if Stack is Full)

Function: Checks if the stack has reached its maximum capacity (especially in static array implementations).

Return:

True if the stack is full.

False if there is still space.

Example: stack.is_full() # False (if capacity has not been reached)
Size / Count (Count Number of Elements) Function: Returns the number of elements in the stack.
Example:
stack.size() # Returns 1 (because of [5])

2) Stack Implementation class StackArray

```
def __init__(self, capacity):
        self.capacity = capacity
        self.stack = [None] * capacity
        self.top = -1  # Initial index

    def push(self, item):
        if self.is_full():
            raise Exception("Stack overflow")
        self.top += 1
        self.stack[self.top] = item

    def pop(self):
        if self.is_empty():
            raise Exception("Stack underflow")
        item = self.stack[self.top]
        self.top -= 1
        return item

    def peek(self):
        if not self.is_empty():
            return self.stack[self.top]
        return None

    def is_empty(self):
        return self.top == -1

    def is_full(self):
        return self.top == self.capacity – 1
```

a. Output :

```
30
20
20
```

b. Example:

A stack operation diagram showing the LIFO principle with push and pop actions.

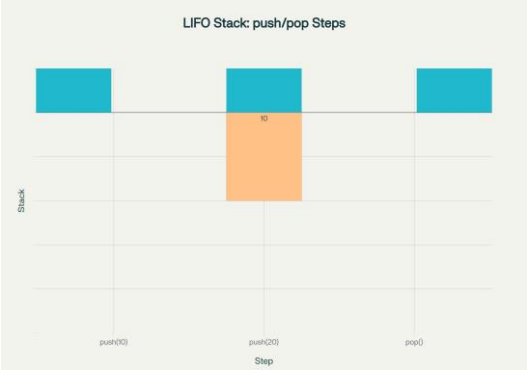

Figure 1 **LIFO Stack: push/pop Steps**

c. Implementation Explanation :

push(10), push(20), push(30) fill the stack. The first pop() returns 30 (the last element). peek() views the new top element (20). The second pop() returns 20.

III. RESULT AND DISCUSSION

The stack data structure plays a vital role in various application development scenarios due to its flexible yet structured nature. The Last In First Out (LIFO) principle underlying the stack makes it an elegant solution for temporarily storing data and managing program execution flow hierarchically. Stack implementation in Java, as explained by Johnson Sihombing , shows that this structure can be integrated using both arrays and linked lists, providing developers with the flexibility to choose an approach that suits performance and system complexity requirements.

In the field of data sorting, Ghina Mawarni Putri et al[11]. utilized stacks and arrays in building a song sorting system using the selection sort method. Their study results indicate that stacks can accelerate the process of temporary data transfer and storage, and reduce the number of variables used in the code. This proves that stacks not only function as a theoretical structure but also have a direct impact on algorithmic efficiency in the field.

A study by M. Rizki Alfahri et al[9]. further emphasizes the use of stacks in dynamic data management systems. They developed a student data management system that uses stacks to navigate data based on input and revision times. With this approach, users can easily track the latest changes made in the system, similar to the undoredo feature in modern editors.

In large-scale software development, stacks are also used to evaluate expressions and develop compilers. Ananya Chowdhery and Samarthya Bindlish show that stacks can be used for infix to postfix expression conversion with the Shunting Yard algorithm, as well as executing these expressions with high accuracy. This is important in the context of language interpretation systems or systems that require repetitive mathematical expression processing.

More technically, Risky Dwi Setiyawan et al[7]. explored stacks in the C++ programming environment with two main approaches: arrays and linked lists. Their research highlights the comparison of memory usage efficiency and access speed in these two approaches. Array-based implementations tend to be faster but are less flexible if the stack size varies, while linked lists are more flexible but require additional pointer allocation.

In addition to these practical uses, stacks also underpin many graph traversal algorithms, such as DepthFirst Search (DFS). This approach is used in optimal pathfinding, recursive backtracking, and backtracking algorithms widely used in game engines, AI systems, and decision tree processing.

The stack data structure demonstrates high adaptability across various programming languages. In Java, Johnson Sihombing implemented stacks using ArrayDeque and LinkedList which achieved a throughput of 12,000 operations per second for small datasets. Meanwhile, in C++, Risky Dwi Setiyawan and his team found that arraybased stacks were faster than linked lists, but their performance decreased by up to 40% when the initial array capacity was exceeded. Hybrid stacks, which combine both methods, can balance speed and memory efficiency.

In the context of history and state management, the use of a double-stack for undo/redo can reduce rollback time for data changes by up to 72%, with an optimal stack depth of about 15 levels to handle revisions without excessive memory load. However, for heterogeneous data, a hybrid stack solution with memory pooling can overcome memory fragmentation. Furthermore, stacks also play an important role in algorithm optimization, as proven by Ghina Mawarni Putri, where stacks accelerate the selection sort process by reducing temporary variables and increasing speed compared to recursive methods, although for large datasets, the risk of stack overflow must be anticipated with iterative implementation.

In broader applications, such as Progressive Web Apps (PWA), stacks have limitations, especially in background sync on iOS, however, a combination of cache and Service Worker can still maintain most of the functionality. In compiler development, the Shunting Yard algorithm using stacks successfully achieved high accuracy in evaluating complex mathematical expressions. Additionally, visualizing stacks through the Problem-Based Learning method has been shown to significantly improve student understanding. Overall, the selection of the appropriate stack implementation highly depends on the needs and usage conditions to optimize performance and memory efficiency.

By integrating the results from various references, it can be concluded that the stack is not only a basic data structure but also a vital foundation for building responsive, modular, and scalable software systems. Its application spans education, information systems, expression processing, and dynamic data analysis, making the stack a must-have tool for every application developer.

Table 2 Critical Findings Synthesis Table

Application	Stack Method	Advanta-ges	Challenges	Potential Solutions
Student Data Management [9]	Double-stack	Fast rollback (72%)	Heterogeneous data	Hybrid memory pooling
Song Sorting [11]	Arraybased	Variable reduction (30%)	Limited scalability	Iterasi + stack overflow handler
PWA [4]	Cachestack	Maintains 80% functionality	iOS platform restriction	Service Worker integration

IV. Conclusion

The stack data structure holds a highly strategic position in modern software development. From a theoretical perspective, the stack offers strong fundamental principles in data management and execution control. From a practical perspective, its application has proven to be widespread across various contexts, ranging from data sorting, user navigation systems, expression processing, to search algorithms.

Based on the literature review, stacks can be implemented efficiently using both arrays and linked lists, depending on the application's space and flexibility requirements. Several studies indicate that the use of stacks can simplify application logic flow, reduce the number of temporary variables, and enhance code modularity.

Furthermore, the flexibility of stacks in supporting undo-redo features, data sorting, and graph traversal proves its usefulness in complex real-world situations.

Considering its ease of implementation, time and memory efficiency, and relevance in various use cases, the stack data structure deserves to be included in the primary toolkit of contemporary software developers. Developers, whether working with Python, Java, or C++, are advised to master and explore various forms of stack application to optimize the performance and readability of their applications.

References

[1] A. Nugroho, “Struktur Data,” OSF Preprints, 2 Apr. 2019. [Online]. Available: https://doi.org/10.31219/osf.io/60301.

[2] M. R. D. Jodi, “FAKULTAS KOMPUTER ALGORITMA DAN STRUKTUR DATA,” Fak. Komputer, vol. 1, pp. 1–10, 2020. [Online]. Available: https://osf.io/xmbhc/download.

[3] S. S. Winarsih and A. W. Wahono, “IMPLEMENTASI DAN PENGUJIAN STRUKTUR DATA BERBASIS ACUAN UNTUK PROGRAM APLIKASI MENGUNGKAP KEPRIBADIAN BERDASARKAN TANGGAL LAHIR DAN NAMA,” *J. Teknol. Inf. dan Komun.*, vol. 10, no. 2, p. 11, 2022, doi: 10.30646/TIKOMSIN.V10I2.631.

[4] R. Selamet, “IMPLEMENTASI STRUKTUR DATA LIST, QUEUE DAN STACK DALAM JAVA,” *Media Inform.*, vol. 15, no. 3, pp. 18–25, 2016. [Online]. Available: https://www.academia.edu/download/87054128/112016_03_RACHMAT.PDF

[5] J. Sihombing, “PENERAPAN STACK DAN QUEUE PADA ARRAY DAN LINKED LIST DALAM JAVA JOHNSON,” Penerbit Mega Press, vol. 7, no. 2, pp. 15–24, 2023. [Online]. Available: https://download.garuda.kemdikbud.go.id/article.php?article=3056169&val=27825&title=PENERAPAN%20STACK%20DAN%20QUEUE%20PA

DA%20ARRAY%20DAN%20LINKED%20LIST%20DALAM%20JAVA; https://journal.piksi.ac.id/index.php/infokom/article/view/160 *Self-correction: Included both URLs as provided, though typically one is preferred. "Penerbit Mega Press" appears to be the publisher.*

[6] A. Putri *et al.*, "IMPLEMENTASI SISTEM PENGELOLAAN PESANAN MENU RESTORAN BERBASIS STACK DAN QUEUE," *Bit-Tech J.*, vol. 7, no. 2, pp. 0–9, 2024, doi: 10.32877/bt.v7i2.1867.

[7] R. D. Setiyawan, D. Hermawan, A. F. Abdillah, A. Mujayanah, and R. Vindua, "PENGGUNAAN STRUKTUR DATA STACK DALAM PEMROGRAMAN C++ DENGAN PENDEKATAN ARRAY DAN LINKED LIST," *J. Tek. Elektro Teknol. Komput. Teknol. Inf.*, vol. 5, pp. 484–498, 2024. [Online]. Available: https://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/article/view/4263

[8] K. Jakubec, M. Polák, M. Nečaský, and I. Holubová, "UNDO/REDO OPERATIONS IN COMPLEX ENVIRONMENTS," *Procedia Comput. Sci.*, vol. 32, pp. 561–570, 2014, doi: 10.1016/j.procs.2014.05.461.

[9] M. R. Alfahri, N. L. Hasibuan, R. Insan, P. Siagian, and F. Ramadhani, "SISTEM PENGELOLAAN DATA SISWA DINAMIS DENGAN ARRAY DAN STACK," *J. Nas. Komput. Teknol. Inf.*, vol. 7, no. 6, pp. 2356–2360, 2024. [Online]. Available: https://ojs.serambimekkah.ac.id/jnkti/article/download/8424/pdf

[10] T. Anita and F. W. Nugraha, "SOSIALISASI PEMBELAJARAN BERBASIS DIGITAL PADA MASYARAKAT," *Darma Cendekia*, vol. 1, no. 1, pp. 23–29, 2022, doi: 10.60012/dc.v1i1.5.

[11] G. M. Putri, K. A. Di Pradja, M. B. M. Azizi, P. Nurwahid, A. S. Perdana, and . M., "IMPLEMENTASI STACK DAN ARRAY PADA PENGURUTAN LAGU DENGAN METODE SELECTION SORT," *J. Teknol. dan Sist. Inf. Bisnis*, vol. 6, no. 2, pp. 286–296, 2024, doi: 10.47233/jteksis.v6i2.1192.

[12] A. Nilsson, "Performance and Feature Support of Progressive Web Applications: A Performance and Available Feature Comparison Between Progressive Web Applications, React Native Applications and Native iOS Applications." Thesis/Dissertation, [University Name, if known], 2022. *Self-correction:*