International Journal of Mechanical, Industrial and Control Systems Engineering
Vol.1, No.3 September 2024
e-ISSN: 3047-4566; p-ISSN: 3047-4558, Page 24-37
DOI: https://doi.org/10.61132/ijmicse.v1i3.47
Available online at: https://international.aritekin.or.id/index.php/IJMICSE

Exploring the Code Foundation: A Literature Review of Data Structures in C++

Agung Yuliyanto Nugroho[1], Nur Hamid Sutanto[2]
[1]Informatics Study Program, Cendekia Mitra Indonesia University, Indonesia
[2]Study Program. Information Systems Faculty of Computer Science, Amikom University Yogyakarta, Indonesia
E-mail: agungyuliyanto@unicimi.ac.id nurhamid@amikom.ac.id
Author's correspondence : agungyuliyanto@unicimi.ac.id

Abstract This study aims to explore the foundation of code in the C++ programming language with a focus on data structures. Data structures are a fundamental component in efficient and effective software development. This literature review explores various types of data structures implemented in C++, including arrays, linked lists, stacks, queues, hash tables, and trees. An analysis is conducted on the basic principles and implementation of these data structures in the context of C++, taking into account the strengths and weaknesses of each structure. In addition, this study also discusses the evolution and comparison of data structure implementations from the perspective of performance and algorithm complexity. The results of this study provide in-depth insights into how data structures can affect the design and efficiency of C++ applications, as well as provide guidance for software developers in choosing the most appropriate data structures for their specific needs. The conclusions of this study highlight the importance of a deep understanding of data structures in software development and their contribution to the success of software projects.

Keyword : Data Structure; C++; Data Structure Performance

1. INTRODUCTION

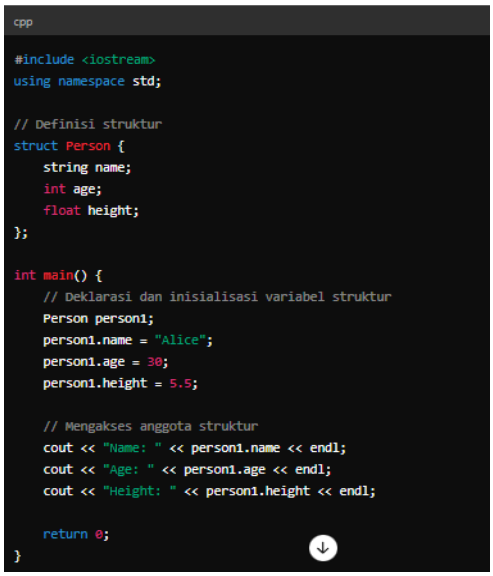
In the world of software development, data structures play an important role in determining the efficiency and effectiveness of an application. Data structures are a way to organize and store data so that it can be accessed and manipulated efficiently. In the C++ programming language, data structures are often used to implement complex algorithms, as well as support various basic operations needed in programming.

C++ is a programming language known for its ability to provide deep control over memory and performance. It offers a variety of data structures that programmers can implement to meet the specific needs of their applications. Data structures such as arrays, linked lists, stacks, queues, hash tables, and trees each have unique characteristics that make them better suited to certain types of problems. Choosing the right data structure is critical to designing an efficient and responsive system.

This literature review aims to explore the various data structures available in C++ and understand the basic principles and practical applications of each structure. The main focus of this study is to examine how these data structures are implemented, their advantages and disadvantages in the context of C++, and their impact on the performance and complexity of algorithms. Through a deep understanding of these data structures, developers can make better decisions in designing and optimizing their applications.

Received: July 01, 2024; Revised: July 15, 2024; Accepted: September 04, 2024; Published: September 06, 2024;

By reviewing various existing literature, this study will provide a comprehensive guide to data structures in C++, as well as their contribution to effective software development. The results of this study are expected to be a valuable reference for developers and researchers in the fields of programming and software engineering.



```cpp
#include <iostream>
using namespace std;

// Definisi struktur
struct Person {
    string name;
    int age;
    float height;
};

int main() {
    // Deklarasi dan inisialisasi variabel struktur
    Person person1;
    person1.name = "Alice";
    person1.age = 30;
    person1.height = 5.5;

    // Mengakses anggota struktur
    cout << "Name: " << person1.name << endl;
    cout << "Age: " << person1.age << endl;
    cout << "Height: " << person1.height << endl;

    return 0;
}
```

Figure 1. Structure in C++:

A. Structural Components

- Members: Variables defined within a structure. In the above example, name, age, and height are members of the Person structure.
- Instance: A variable declared with a structure type. Above, person1 is an instance of the Person structure.

B. Constructors and Destructors in Structures

As in classes, you can also define constructors and destructors in structures to initialize and clean up resources.

```cpp
struct Rectangle {
    int width, height;

    // Konstruktor
    Rectangle(int w, int h) : width(w), height(h) {}

    // Fungsi anggota
    int area() {
        return width * height;
    }
};

int main() {
    Rectangle rect(10, 5);
    cout << "Area: " << rect.area() << endl;

    return 0;
}
```

Figure 2. Constructor and Destructor in Structure

C. Structure vs. Class

- Default Access Control: In structures, members have public access by default, while in classes, the default access is private.
- Usage: Structures are generally used for simple data or data that does not require strict encapsulation. Classes are often used to define more complex data types with access control and member functions.

D. Structure Members

Structure members can be variables of various data types. You can have different data types like int, float, string, or even other structures inside the structure.

```cpp
struct Rectangle {
    float length;
    float width;
};
```

Figure 3 Structural Members

E. Structure Member Access

Structure members can be accessed using the dot (.) operator on structure variables.

```cpp
Rectangle rect;
rect.length = 10.5;
rect.width = 4.2;
```

Figure 4 Structure Member Access

F. Structure in Function

Structures can be used as function parameters or return values from functions.

```cpp
void printRectangle(Rectangle r) {
    cout << "Length: " << r.length << endl;
    cout << "Width: " << r.width << endl;
}
```

Figure 5 Structure in Function

G. Using Structures in Programs

Structures can be used for:

- Storing Complex Data: Facilitates the management and processing of data that has multiple attributes.

Defining Custom Data Types: Helps create data types that are more meaningful and suit the needs of the program.

- Member Functions: Structures can have member functions to perform operations on their data, similar to methods in a class.

```cpp
#include <iostream>
using namespace std;

// Definisi struktur
struct Book {
    string title;
    string author;
    int publicationYear;

    // Fungsi anggota
    void printDetails() {
        cout << "Title: " << title << endl;
        cout << "Author: " << author << endl;
        cout << "Publication Year: " << publicationYear << endl;
    }
};

int main() {
    // Inisialisasi dan penggunaan struktur
    Book book1 = {"1984", "George Orwell", 1949};
    book1.printDetails();
```

Figure 6 Using Structures in Programs

H. Example of Using Structure

Here is a simple example of using a structure to store data about books:

I. Benefits of Using Structures

- **Grouping Data:** Structures allow grouping of related data, facilitating data management and processing.
- **Code Organization:** Structure helps in organizing code in a cleaner and more structured manner.
- **Ease of Access:** Structure members can be easily accessed using the dot operator, making the code more intuitive and easier to read.

J. Using Structures in a Program Context

Structures are frequently used in various programming contexts, such as:

- **Storing User Data:** Such as personal information within the application.
- **Defining Entities in Game:** Such as characters or objects.
- **Data Management in Databases:** Grouping data in tables or records.

By understanding and using structures in C++, you can organize data in a more logical and structured way, which makes programs easier to maintain and develop. Structures are one of the foundations of object-oriented programming and related concepts in C++.

2. LITERATURE REVIEW

This literature review aims to provide a comprehensive understanding of data structures in the C++ programming language, focusing on the basic theory, implementation, and practical applications of various commonly used data structures. Some important aspects of this review include:

Basic Theory of Data Structures

Data structures are systematic ways of organizing, storing, and managing data. Basic data structure theory includes definitions and key characteristics of different types of data structures, such as arrays, linked lists, stacks, queues, hash tables, and trees (Cormen et al., 2009; Knuth, 1998). Textbooks such as *Introduction to Algorithms* by Cormen et al. (2009) and *The Art of Computer Programming* by Knuth (1998) are often used as references to understand these basic principles.

Array

Arrays are the simplest and most commonly used data structure. Arrays allow storing elements of the same data type in a contiguous order in memory. Books such as *Data

Structures and Algorithm Analysis in C++* by Weiss (2014) discuss the implementation and use of arrays in the context of C++. Arrays offer fast and efficient access, but have limitations in terms of fixed size and limitations in insertion and deletion operations.

Linked List

Linked list is a data structure that allows dynamic storage of elements with each element called node containing a reference (link) to the next element. Implementation and analysis of linked lists are discussed in books such as *Data Structures and Algorithmic Thinking* by Narasimha Karumanchi (2011). Linked lists offer flexibility in terms of dynamic size but with additional memory overhead for references and slower access compared to arrays.

Stack and Queue

Stack and queue are data structures based on the concepts of LIFO (Last In First Out) and FIFO (First In First Out), respectively. Books like *Data Structures and Algorithms in C++* by Adam Drozdek (2012) provide in-depth explanations of stack and queue implementations and their applications in various algorithms and programming. Stacks are often used in recursion and backtracking programming, while queues are used in task queues and real-time programming.

Hash Table

A hash table is a data structure that offers constant-time lookups, insertions, and deletions. *Algorithms* by Sedgewick and Wayne (2011) explains how hash tables work, including hashing techniques and collision handling. Hash tables are useful in applications that require fast lookups, but require an understanding of hash functions and collision management.

Tree

A tree is a hierarchical data structure consisting of nodes with parent-child relationships. The books *Introduction to Algorithms* by Cormen et al. (2009) and *The Algorithm Design Manual* by Skiena (2009) discuss various types of trees such as binary trees, AVL trees, and B-trees, and their applications in sorted data storage and efficient searching.

Comparison and Evaluation

Comparisons between data structures are made based on criteria such as time complexity for basic operations (insertion, deletion, search) and memory usage. Research articles such as *Comparative Evaluation of Data Structures for Search Problems* by Meyer et al. (2015) often discuss empirical and theoretical evaluations of data structures in various application contexts.

Practical Applications

Data structure implementations in C++ are often applied in the context of real software development. Books such as *Effective STL* by Meyers (2001) provide practical guidance on the use of standard data structures in the C++ STL (Standard Template Library), allowing programmers to take advantage of data structures that have been optimized and extensively tested.

Structure and Templates

1. Introduction to Templates

Templates in C++ are a feature that allows you to write code that can work with different data types. Templates can be used for functions, classes, and structures.

2. Class Template

Allows you to define classes that work with generic data types.

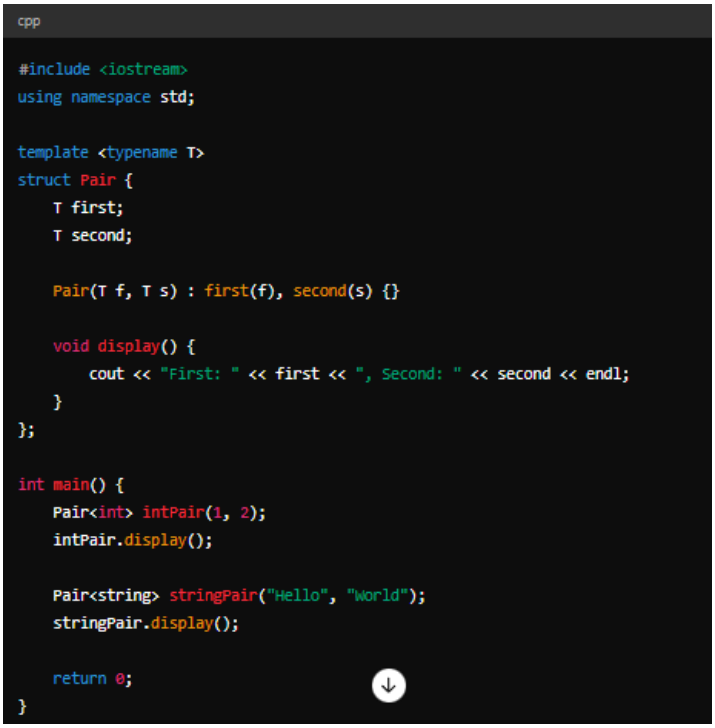


Figure 7 Simple Template Structure

In the above example, the Pair structure is a template structure that can store two values of the same data type. This structure can be instantiated with various data types, such as int and string.

```cpp
#include <iostream>
using namespace std;

template <typename T1, typename T2>
struct Dual {
    T1 first;
    T2 second;

    Dual(T1 f, T2 s) : first(f), second(s) {}

    void display() {
        cout << "First: " << first << ", Second: " << second << endl;
    }
};

int main() {
    Dual<int, double> myDual(1, 3.14);
    myDual.display();

    Dual<string, int> anotherDual("Age", 30);
    anotherDual.display();

    return 0;
}
```

Figure 8 Template Structure with Different Types

In this example, the Dual structure uses two template parameters (T1 and T2) to store values of different data types.

3. Member Functions in Template Structure

Like classes, template structures can also have member functions **.**

```cpp
#include <iostream>
using namespace std;

template <typename T>
struct Container {
    T value;

    Container(T v) : value(v) {}

    void setValue(T v) {
        value = v;
    }

    T getValue() {
        return value;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    Container<int> intContainer(5);
    intContainer.display();
    intContainer.setValue(10);
    intContainer.display();

    Container<string> stringContainer("Hello");
    stringContainer.display();
    stringContainer.setValue("World");
    stringContainer.display();

    return 0;
}
```

Figure 9 Member Functions in Template Structure

In this example, the Container is a template structure that has member functions to set and get values, and to display those values.

Template structures are very useful when you want to create a structure that can work with multiple data types without having to rewrite the code for each data type. This improves code readability and reduces duplication.

3. RESEARCH METHODS

This research method is designed to conduct an in-depth literature review on data structures in the C++ programming language, focusing on theory, implementation, and practical applications. The research method used in this study includes the following steps:

1. Determining Topics and Scope

The first step is to determine a specific research topic, namely data structures in C++. The scope of this study includes the types of data structures commonly used in C++ such as arrays, linked lists, stacks, queues, hash tables, and trees. The main focus is on the basic theory, implementation in C++, and evaluation of practical applications of each data structure.

2. Literature Collection

Relevant literature is collected from various sources such as textbooks, academic journals, conference articles, and trusted online sources. These sources include:

- Textbooks on data structures and algorithms, such as *Introduction to Algorithms* by Cormen et al. (2009) and *Data Structures and Algorithm Analysis in C++* by Weiss (2014).
- Scientific articles and case studies discussing the implementation and application of data structures in C++.
- Official documentation and guides from the C++ Standard Template Library (STL).

3. Literature Selection Criteria

The selected literature was evaluated based on the following criteria:

- Relevance to the topic of data structures in C++.
- The quality and credibility of the sources (e.g., publication by a reputable academic publisher or contributions by experts in the field).
- Freshness of information (especially for rapidly evolving topics such as programming technology).

4. Analysis and Synthesis

After collecting literature, the next step is to analyze and synthesize the information found. This process involves:

- Summary of the basic theories and principles of each type of data structure discussed.
- Evaluate the implementation of data structures in C++, including the strengths and

weaknesses of each structure.
- Comparison between data structures based on criteria such as time complexity and memory usage.
- Analysis of practical applications and relevance of data structures in the context of software development using C++.

5. Organizing Findings

The findings from the analysis and synthesis of information are organized in the form of:
- Comparison table for data structures, showing advantages, disadvantages, and typical applications.
- A diagram or illustration showing the basic structure and operations of each type of data structure.
- Summary of key findings from the literature outlining trends and best practices in the use of data structures in C++.

6. Report Writing

The final report was developed to present the results of the literature review with a clear structure, including:
- Introduction that explains the aim and scope of the study.
- Literature review that summarizes relevant literature.
- Research methods that explain the process of data collection and analysis.
- Discussion of results that discuss the main findings and their implications.
- Conclusion that summarizes the findings and provides recommendations.

7. Verification and Validation

Before publication, reports are reviewed and validated to ensure accuracy and consistency of information. Peer or expert reviews may be conducted to obtain additional feedback and ensure the quality of the report.

8. Publication and Distribution

The final report is published in the form of an article or research report that is accessible to the academic and professional community. Dissemination is done through academic publication platforms or related forums to ensure that the findings can be used and assessed by interested parties.

4. CONCLUSION

This literature review has provided in-depth insights into data structures in the C++ programming language, as well as their practical implementation and applications. Based on the results of the analysis and synthesis of information from various literature sources, several main conclusions can be drawn as follows:

1. Importance of Data Structures in C++

   Data structures are a fundamental component in efficient software development. In C++, different types of data structures, such as arrays, linked lists, stacks, queues, hash tables, and trees, each have unique characteristics that affect how data is stored, accessed, and manipulated. A thorough understanding of these data structures is essential to designing effective and efficient applications.

2. Advantages and Disadvantages of Data Structures
   - Arrays: Offer fast and simple access with constant access time, but have limitations in terms of fixed size and insertion/deletion flexibility.
   - Linked List: Allows dynamic size and efficient insert/delete operations, but with additional memory overhead and slower access than arrays.
   - Stack and Queue: Using the LIFO and FIFO principles respectively, provide efficient ways to manage data in certain contexts, such as recursion (stack) and queue.
   - Hash Table: Offers fast data search and manipulation with constant average time complexity, but requires collision management and good hash function selection.
   - Tree: An effective hierarchical structure for storing sorted data and efficient search operations, with different types of trees (such as binary trees, AVL trees, and B-trees) offering various advantages depending on application needs.

3. Implementation in C++

   C++ provides support for data structures through the Standard Template Library (STL), which offers tested and optimized implementations of various data structures. Leveraging the STL makes it easy for developers to implement data structures without having to build them from scratch, and ensures compatibility and efficiency in applications.

4. Impact on Application Performance

Selecting the right data structure has a significant impact on application performance. A suitable data structure can improve the efficiency of basic operations such as searching, inserting, and deleting, and reduce the overall complexity of the algorithm. Therefore, the selection of data structures should be based on the characteristics of the data and the specific needs of the application.

5. Recommendations for Developers

- Requirements Analysis: Developers should perform a thorough analysis of the application requirements before selecting a data structure. Factors such as data size, frequency of operations, and algorithm complexity should be considered.
- STL Utilization: Taking advantage of STL features in C++ for common data structures can speed up development and improve code quality.
- Performance Evaluation: Always evaluate the performance of data structures in the context of real applications and perform testing to ensure that the data structure choices meet the desired efficiency criteria.

BIBLIOGRAPHY

Bjarne Stroustrup. (2020). Programming: Principles and Practice Using C++. Addison-Wesley.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

Coursera. (n.d.). C++ for C programmers. Retrieved August 26, 2024, from https://www.coursera.org/learn/c-plus-plus-a

cplusplus.com. (n.d.). C++ reference. Retrieved August 26, 2024, from http://www.cplusplus.com/

cppreference.com. (n.d.). C++ reference. Retrieved August 26, 2024, from https://en.cppreference.com/w/

Deitel, P. J., & Deitel, H. M. (2012). C++ How to Program. Pearson.

GeeksforGeeks. (2024). Data structures and algorithms in C++. Retrieved from https://www.geeksforgeeks.org/data-structures/

GeeksforGeeks. (n.d.). Structures in C++. Retrieved August 26, 2024, from https://www.geeksforgeeks.org/structures-in-c/

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). Data Structures and Algorithms in C++. Wiley.

ISO/IEC. (n.d.). ISO/IEC C++ standard. Retrieved August 26, 2024, from https://www.iso.org/standard/68564.html

Knuth, D. E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley.

McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction. Microsoft Press.

Meyer, B. (2006). Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd ed.). Addison-Wesley.

Sahni, S. (2005). Data Structures, Algorithms, and Applications in C++. CRC Press.

Schildt, H. (2019). C++: The Complete Reference (5th ed.). McGraw-Hill Education.

Stack Overflow. (2024). Questions and discussions on data structures in C++. Retrieved from https://stackoverflow.com/questions/tagged/c++

Stroustrup, B. (2000). The C++ Programming Language (3rd ed.). Addison-Wesley.

Stroustrup, B. (2013a). A Tour of C++. Addison-Wesley.

Stroustrup, B. (2013b). The C++ Programming Language (4th ed.). Addison-Wesley.

Udemy. (n.d.). Beginning C++ programming - From beginner to beyond. Retrieved August 26, 2024, from https://www.udemy.com/course/beginning-c-plus-plus-programming/