JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

PENGGUNAAN STRUKTUR DATA *STACK* DALAM PEMROGRAMAN *C++* DENGAN PENDEKATAN *ARRAY* DAN *LINKED LIST*

Risky Dwi Setiyawan[1], Doni Hermawan[2], Ahmad Fahmi Abdillah[3], Arsil Mujayanah[4], Raditia Vindua[5,]
[1,2,3,4,5] *Fakultas Ilmu Komputer, Universitas Pamulang*
email: riskyydwiss0808@gmail.com[1], donihermawan01@gmail.com[2], adielzbuser1916@gmail.com[3], arsylmuzayanah@gmail.com[4], dosen02380@unpam.ac.id[5]

Informasi Artikel

Riwayat artikel :
Disubmit : 15 Desember 2024
Direvisi : 17 Desember 2024
Diterima : 23 Desember 2024
Dipublikasi : 29 Desember 2024

Keywords:
stack, array, linked list, execution efficiency, memory usage, data structures

ABSTRACT

The background of this research is based on the importance of efficiency in implementing the Stack data structure, which is widely used in various programming applications such as mathematical expression evaluation, memory management, and undo/redo features. Two common approaches, namely Array and Linked List, each have their advantages and disadvantages in terms of execution efficiency, memory usage, and flexibility. This study compares the implementation of the Stack data structure using these two approaches, focusing on execution time efficiency, memory usage, and size flexibility.Using an experimental method, the implementation was conducted in C++ programming language through scenarios such as mathematical expression evaluation and undo/redo features. The results show that the Array approach is more efficient for static data access, while the Linked List excels in size flexibility and dynamic memory allocation for variable data. The study concludes that the choice of approach depends on the specific application requirements. These findings are expected to assist software developers in selecting the appropriate Stack implementation method.

Kata Kunci:
Stack, *Array*, *Linked list*, efisiensi waktu, penggunaan memori, struktur data

ABSTRAK

Latar belakang penelitian ini didasarkan pada pentingnya efisiensi dalam implementasi struktur data Stack, yang banyak digunakan dalam berbagai aplikasi pemrograman seperti evaluasi ekspresi matematika, manajemen memori, dan fitur undo/redo. Dua pendekatan umum, yaitu *Array* dan *Linked List*, memiliki kelebihan dan kekurangan masing-masing dalam hal efisiensi waktu, penggunaan memori, dan fleksibilitas. Penelitian ini membahas perbandingan implementasi struktur data *Stack* menggunakan pendekatan *Array* dan *Linked list* dalam aspek efisiensi waktu eksekusi, penggunaan memori, dan fleksibilitas ukuran. Dengan metode eksperimen, implementasi dilakukan menggunakan bahasa pemrograman C/C++ melalui skenario seperti evaluasi ekspresi matematika dan fitur *undo/redo*. Hasil penelitian menunjukkan bahwa pendekatan *Array* lebih efisien dalam kecepatan akses untuk data statis, sementara *Linked list* unggul dalam fleksibilitas ukuran dan alokasi memori dinamis untuk data yang bervariasi. Studi ini menyimpulkan bahwa pemilihan pendekatan bergantung pada kebutuhan spesifik aplikasi. Temuan ini diharapkan dapat membantu pengembang perangkat lunak dalam memilih metode implementasi *Stack* yang tepat.

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

PENDAHULUAN

Di zaman modern, struktur data memainkan kiprah krusial pada pengembangan perangkat lunak. Struktur data yang sangat penting adalah *Stack,* Menurut (Trivusi, 2022) *Stack* atau pada bahasa Indonesia diartikan tumpukan, Merupakan struktur data linier yang mengikuti prinsip *Last In First Out* (LIFO). Artinya elemen yang terakhir disisipkan akan menjadi elemen pertama yang keluar.Sependapat dengan (Purwono et al., 2023) Cara kerjanya seperti ada sebuah tumpukan baju di dalam lemari dan baju yang paling atas (*Last In*) terlebih dahulu kita ambil dan kenakan *(First Out*). yang banyak digunakan dalam berbagai aplikasi seperti evaluasi rumus, manajemen memori, dan pemrosesan fungsi secara rekursi.

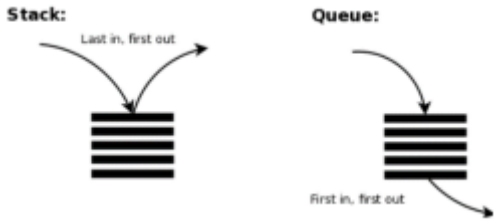
Melalui analisis komprehensif, kami mempertimbangkan kekuatan dan kelemahan masing-masing pendekatan, serta situasi di mana pendekatan yang satu mungkin lebih baik dibandingkan pendekatan lainnya.Menurut (Hafiz et al., 2023) struktur data *Stack* sebagai proses penumpukan nampan di suatu kafetaria. Setiap nampan baru akan ditumpukkan di atas tumpukan nampan yang sudah ada dan nampan terakhir di dalam tumpukan merupakan nampan pertama yang dapat diambil dari tumpukan tersebut. Penggunaan *Stack* banyak digunakan dalam pemrograman dan algoritma, mulai dari manajemen memori hingga penanganan eksekusi fungsi

Pengertian *Stack* Menurut (Soffya Ranti, 2022) *Stack* adalah struktur data linier yang mengikuti aturan tertentu untuk melakukan operasi. Menambah dan menghapus elemen dari tumpukan hanya dapat terjadi di satu lokasi, yaitu ujung tumpukan. *Stack* bersifat LIFO (*Last in First Out*) dan objek yang terakhir masuk ke dalam *Stack* akan menjadi benda pertama yang dikeluarkan dari *Stack* itu. (Sihombing et al., 2020)

Menurut (Setiawan, n.d.)*Stack* dikenal sangat luas pemakaiannya dalam menyelesaikan berbagai macam problema. Kompiler (*Compiler*), Sistem Operasi (*Operating System*) dan berbagai Program Aplikasi (*Application Program*) banyak menggunakan konsep *Stack* tersebut. Salah satu contoh adalah problema Penjodohan Tanda Kurung (*Matching Parantheses*). Misalnya, dalam algoritma pencarian dan pengurutan,

Stack memiliki beberapa fungsi dan operasi antara lain: (Geeksforgeeks, 2024b) Operasi utama pada *Stack* meliputi: 1) Push, yaitu menambahkan elemen baru ke dalam *Stack* pada posisi teratas. 2) Pop, yaitu menghapus elemen teratas dari *Stack* dan mengembalikan nilainya. 3) Peek atau top, yaitu mengecek elemen teratas dari *Stack* tanpa menghapusnya. 4) Isempty, yaitu memeriksa apakah *Stack* kosong atau tidak. 5) Size, yaitu mengembalikan jumlah elemen dalam *Stack*. Fungsi pada *Stack* antara lain: Mengorganisasi data dan memudahkan penyimpanan komputer , Digunakan dalam berbagai algoritma, seperti *tower of hanoi, tree traversal*, dan rekursi

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM



Gambar 1. Ilustrasi *Stack*

Pengertian *Array* menurut (Indah Kusuma Astuti, 2019) Larik (Bahasa Inggris*: Array*), dalam ilmu komputer, adalah suatu tipe data terstruktur yang dapat menyimpan banyak data dengan suatu nama yang sama dan menempati tempat, di memori yang berurutan (kontinu) serta bertipe data sama , Elemen-elemen dalam *Array* diakses menggunakan indeks, yang dimulai dari 0 untuk elemen pertama. Sependapat dengan hal Tersebut (Putri et al., 2024)*Array* adalah struktur data variabel untuk menyimpan sekumpulan data dengan tipe data yang sama.

Gambar 2 Visualisasi deklarasi *Array*

Array berdimensi satu Merupakan sekumpulan item data yang ditunjukkan atau diacu oleh satu identifier dan disusun menjadi rangkaian. Menurut (Melati, 2024) Sebuah *Array* 1 dimensi bekerja seolah-olah seperti jalur tunggal ini. Dalam memori komputer, *Array* ini adalah kumpulan elemen data yang diurutkan secara berurutan dalam satu dimensi. Dalam contoh ini, nilai = (3 6 8 1 5).

Gambar 3 Ilustrasi *Array* 1 dimensi

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297

Array berdimensi dua *Array* berdimensi 2 adalah suatu kumpulan data terstruktur yang terdiri dari dua dimensi atau dua indeks, yaitu baris dan kolom. Menurut (Triase, 2020) *Array* berdimensi 2 dapat digunakan untuk menyimpan data yang terstruktur dalam bentuk matriks atau tabel. Deklarasi *Array* berdimensi 2 dilakukan dengan menentukan tipe data elemen yang disimpan, nama variabel *Array*, dan ukuran *Array* pada setiap dimensi.

Gambar 4 Visualisasi *Array* 2 dimensi

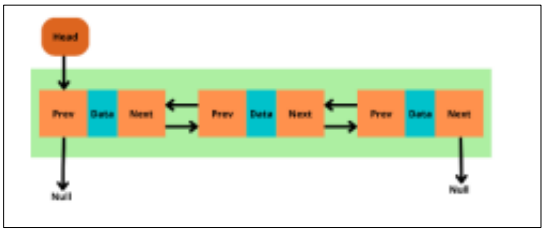
Pengertian *Linked list*, menurut (Sihombing et al., 2020) Linked list adalah suatu bentuk struktur data yang berupa sekumpulan elemen data yang bertipe sama dimana tiap elemen saling berkaita atau dihubungkan dengan elemen lain melalui suatu pointer ,Pointer adalah alamat elemen data yang tersimpan di memori. Mengacu elemen dengan pointer membuat elemen bersebelahan secara logik, meskipun *Linked list* menyimpan elemen di lokasi memori yang tidak teratur, dengan setiap elemen memiliki referensi ke elemen berikutnya(Mustakim et al., 2024). Ada beberapa jenis *Linked list* yang dapat di proses : singly *Linked list*, doubly *Linked list*, singly circular *Linked list*, doubly circular *Linked list*

1. Singly *Linked list* Menurut (Joko Musridho, 2024) (Ahmad, 2024)Single *Linked list* adalah Daftar terhubung yang setiap simpul pembentuknya mempunyai satu rantai(link) ke simpul lainnya. simpul yang saling terhubung satu sama lain dengan menggunakan pointer. Setiap simpul dalam singly *Linked list* memiliki dua bagian, yaitu data dan pointer yang menunjuk ke simpul berikutnya. Singly *Linked list* hanya memiliki satu arah, yaitu dari simpul awal (head) ke simpul akhir (tail).

Gambar 5 Ilustrasi singly *Linked list*

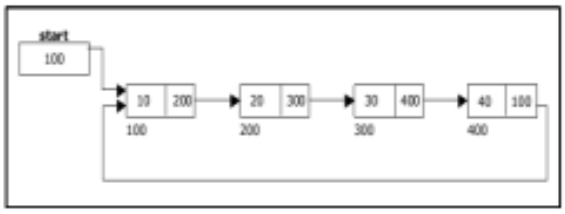
JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

2. *Doubly Linked list* Doubly *Linked list* adalah suatu bentuk struktur data yang terdiri dari simpul-simpul yang saling terhubung satu sama lain dengan menggunakan dua pointer, memungkinkan traversal daftar yang efisien di kedua arah. Hal ini karena setiap simpul dalam daftar berisi pointer ke simpul sebelumnya dan pointer ke simpul berikutnya. Hal ini memungkinkan penyisipan dan penghapusan simpul dari daftar dengan cepat dan mudah, serta traversal daftar yang efisien di kedua arah(geeksforgeeks, 2024a).



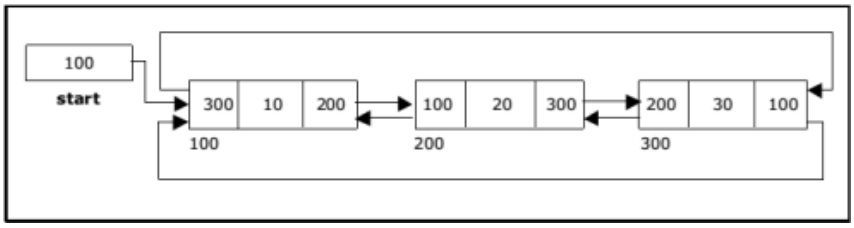
Gambar 6 Ilustrasi doubly *Linked list*

3. Singly circular *Linked list* (Ayoosh Sharma, 2021)mengatakan Dalam variasi ini, simpul terakhir dari daftar menunjuk ke simpul pertama dari daftar dan karenanya membentuk sebuah lingkaran. Setiap node dalam daftar tertaut melingkar terdiri dari dua bagian: data dan penunjuk ke node berikutnya.



Gambar 7 Ilustrasi circular single *Linked list*

4. Doubly circular *Linked list* (alphacodingskills, n.d.)Mengatakan Doubly circular *Linked list* adalah struktur data linier, yang elemen-elemennya disimpan dalam bentuk simpul. Setiap simpul berisi tiga sub-elemen. Bagian data yang menyimpan nilai elemen, bagian sebelumnya yang menyimpan penunjuk ke simpul sebelumnya, dan bagian berikutnya yang menyimpan penunjuk ke simpul berikutnya.



Gambar 8 Ilustrasi doubly circular *Linked list*

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

METODE PENELITIAN

Penelitian ini merupakan jenis penelitian eksperimental yang berfokus pada analisis kinerja struktur data *Stack* menggunakan dua pendekatan, yaitu *Array* dan *Linked list*. Penelitian dilakukan di lingkungan pemrograman C++ dengan simulasi studi kasus pada evaluasi ekspresi matematika dan implementasi undo/redo.

Untuk mengukur kinerja kedua pendekatan, penelitian ini menggunakan pendekatan kuantitatif dengan tiga variabel utama. Pertama, waktu eksekusi untuk operasi dasar *Stack* seperti *push* dan *pop* dianalisis untuk mengetahui efisiensi waktu dari masing-masing metode. Kedua, penggunaan memori dievaluasi untuk mengukur sejauh mana masing-masing pendekatan memanfaatkan memori yang tersedia. Ketiga, fleksibilitas ukuran dianalisis untuk menilai kemampuan pendekatan dalam menyesuaikan ukuran *Stack* sesuai kebutuhan program.

Hipotesis penelitian yang diuji adalah bahwa pendekatan *Array* memiliki keunggulan dalam kecepatan akses dibandingkan *Linked list*, sementara pendekatan *Linked list* lebih unggul dalam fleksibilitas pengelolaan ukuran data dibandingkan *Array*. Untuk menguji hipotesis ini, penelitian dilakukan dengan langkah-langkah sistematis, mulai dari implementasi struktur data, pengujian skenario, hingga analisis data.

Tahapan penelitian dimulai dengan identifikasi masalah dan kebutuhan yang menjadi dasar perancangan sistem. Struktur data *Stack* diimplementasikan secara terpisah menggunakan *Array* dan *Linked list*, di mana setiap pendekatan dirancang untuk mendukung operasi dasar seperti *push*, *pop*, dan *isEmpty*. Selanjutnya, dilakukan pengujian dengan dua skenario utama. Pada skenario pertama, *Stack* berbasis *Array* digunakan untuk mengonversi ekspresi matematika dari notasi infix ke postfix. Sementara itu, pada skenario kedua, *Stack* berbasis *Linked list* diterapkan untuk mendukung fitur undo/redo pada aplikasi sederhana.

Hasil dari pengujian tersebut kemudian dikumpulkan, meliputi data terkait waktu eksekusi, penggunaan memori, dan fleksibilitas ukuran *Stack*. Data yang diperoleh dianalisis secara kuantitatif untuk mengevaluasi performa masing-masing pendekatan dalam berbagai skenario. Analisis ini dilakukan dengan bantuan alat pemrograman yang relevan untuk memastikan akurasi hasil. Hasil analisis disusun untuk mempermudah pemahaman, kemudian disertai pembahasan mendalam yang mengidentifikasi kelebihan dan kekurangan dari setiap pendekatan.

Dengan pendekatan ini, penelitian diharapkan dapat memberikan wawasan yang lebih mendalam tentang implementasi *Stack* menggunakan *Array* dan *Linked list*, serta membantu pengembang perangkat lunak dalam memilih metode yang paling sesuai berdasarkan kebutuhan spesifik aplikasi mereka.

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

Implementasi Studi Kasus

Kasus 1: evaluasi ekspresi matematika infix menggunakan *Stack* (pendekatan *Array*)

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#define max_size 100
typedef struct {
    char data[max_size];
    int top;
} StackArray;
void push(StackArray *Stack, char element) {
    if (Stack->top < max_size - 1) {
        Stack->top++;
        Stack->data[Stack->top] = element;
    } else {
        printf("Stack overflow\n");
        exit(1);
    }
}
char pop(StackArray *Stack) {
    char element = '\0';
    if (Stack->top >= 0) {
        element = Stack->data[Stack->top];
        Stack->top--;
    } else {
```

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

```
        printf("Stack underflow\n");
        exit(1);
    }
    return element;
}
int isoperator(char symbol) {
    return (symbol == '+' || symbol == '-' || symbol == '*' || symbol == '/');
}
int precedence(char operator) {
    switch (operator) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        default:
            return 0;
    }
}
void infixtopostfix(char infix[], char postfix[]) {
    StackArray Stack;
    Stack.top = -1;
    int i = 0, j = 0;
    while (infix[i] != '\0' && infix[i] != '\n') {
```

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

```
        char symbol = infix[i];
        if (isoperator(symbol)) {
            while (Stack.top >= 0 && precedence(Stack.data[Stack.top]) >= precedence(symbol)) {
                postfix[j++] = pop(&Stack);
            }
            push(&Stack, symbol);
        } else if (symbol == '(') {
            push(&Stack, symbol);
        } else if (symbol == ')') {
            while (Stack.top >= 0 && Stack.data[Stack.top] != '(') {
                postfix[j++] = pop(&Stack);
            }
            pop(&Stack);  // Remove '(' from Stack
        } else {
            postfix[j++] = symbol;
        }
        i++;
    }
    while (Stack.top >= 0) {
        postfix[j++] = pop(&Stack);
    }
    postfix[j] = '\0';
}
int main() {
    char infix[max_size], postfix[max_size];
```

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

```
    printf("Masukkan ekspresi infix: ");
    fgets(infix, sizeof(infix), stdin);
    infixtopostfix(infix, postfix);
    printf("Ekspresi postfix: %s\n", postfix);
    return 0;
}
```

Dalam ekspresi matematika infix, kita dapat menggunakan *Stack* (pendekatan *Array*) untuk mengonversi ekspresi tersebut menjadi bentuk postfix atau menghitung hasilnya. Berikut adalah contoh sederhana menggunakan *Stack Array* untuk menghitung ekspresi infix:

kasus 2: implementasi undo/redo menggunakan *Stack* (pendekatan *Linked list*)

Dalam aplikasi pengolah teks atau editor, kita dapat menggunakan *Stack* (pendekatan *Linked list*) untuk mendukung operasi undo dan redo. Berikut adalah contoh sederhana menggunakan *Stack Linked list* untuk implementasi undo/redo:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    char data;
    struct node *next;
} node;
typedef struct {
    node *top;
} Stacklinkedlist;
void push(Stacklinkedlist *Stack, char element) {
    node *newnode = (node *)malloc(sizeof(node));
    if (newnode == NULL) {
        printf("Memory allocation error\n");
        exit(1);
    }
    newnode->data = element;
    newnode->next = Stack->top;
    Stack->top = newnode;
```

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

```
}
char pop(Stacklinkedlist *Stack) {
    char element = '\0';
    if (Stack->top != NULL) {
        node *temp = Stack->top;
        element = temp->data;
        Stack->top = temp->next;
        free(temp);
    } else {
        printf("Stack underflow\n");
        exit(1);
    }
    return element;
}
void printStack(Stacklinkedlist *Stack) {
    node *current = Stack->top;
    while (current != NULL) {
        printf("%c ", current->data);
        current = current->next;
    }
    printf("\n");
}
int main() {
    Stacklinkedlist undoStack;
    undoStack.top = NULL;
    Stacklinkedlist redoStack;
    redoStack.top = NULL;
    // Simulate user actions
    push(&undoStack, 'a');
    push(&undoStack, 'b');
    push(&undoStack, 'c');
    printf("Undo Stack: ");
    printStack(&undoStack);
```

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

```
    // Perform undo operation
    char undoelement = pop(&undoStack);
    push(&redoStack, undoelement);
    printf("After undo:\n");
    printf("Undo Stack: ");
    printStack(&undoStack);
    printf("Redo Stack: ");
    printStack(&redoStack);
    return 0;
}
```

Dalam kedua contoh ini, *Stack Array* digunakan untuk mengonversi ekspresi matematika infix menjadi postfix, sedangkan *Stack Linked list* digunakan untuk implementasi undo/redo dalam pengolah teks. Perhatikan bahwa contoh-contoh ini sederhana dan digunakan untuk menjelaskan konsep penggunaan *Stack* dalam pemrograman

HASIL DAN PEMBAHASAN

Perbandingan kinerja

Menurut pendapat (Sihombing et al., 2020) pendekatan *Array* dan *Linked list* adalah dua cara umum untuk mengimplementasikan *Stack*. Berikut adalah perbandingan kinerja penggunaan *Stack* dengan pendekatan *Array* dan *Linked list* dalam beberapa aspek: a. Alokasi memori: *Array*: membutuhkan alokasi memori statis pada saat deklarasi. Ukuran *Array* harus ditentukan sebelumnya, dan perubahan ukuran dapat memerlukan pembuatan *Array* baru dan pengalihan data. *Linked list*: memungkinkan alokasi memori dinamis, yang berarti ukuran *Stack* dapat bertambah atau berkurang secara dinamis tanpa harus menentukan ukuran sebelumnya. B. Penyisipan dan penghapusan elemen: *Array*: penyisipan dan penghapusan elemen di tengah *Array* (bukan di ujung) dapat memerlukan pergeseran elemen, yang dapat memakan waktu pada *Stack* yang besar. *Linked list*: penyisipan dan penghapusan elemen di tengah *Stack* dilakukan dengan mengubah tautan antar simpul, yang lebih efisien daripada pergeseran elemen pada *Array*. C. Akses ke elemen tertentu: *Array*: akses ke elemen tertentu dalam *Array* dilakukan secara langsung melalui indeks. Proses ini memiliki kompleksitas waktu (1). *Linked list*: akses ke elemen tertentu dalam *Linked list* memerlukan penelusuran dari awal hingga elemen yang diinginkan, yang memiliki kompleksitas waktu o (n).

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

Kelebihan dan kekurangan

Array: Kelebihan: 1. Akses langsung: memungkinkan akses langsung ke elemen menggunakan indeks, sehingga pencarian dan akses elemen memiliki kompleksitas waktu o(1). 2. Implementasi sederhana: implementasinya lebih sederhana dan membutuhkan overhead yang lebih rendah karena tidak ada tautan tambahan atau alokasi memori dinamis. Kekurangan: 1. Ukuran tetap: memiliki ukuran tetap yang ditentukan pada saat deklarasi, sulit untuk mengubah ukuran *Stack* secara dinamis tanpa pembuatan *Array* baru. 2. Fragmentasi memori: mungkin mengalami fragmentasi memori karena ukuran *Array* tetap sepanjang waktu program berjalan. 3. Pergeseran elemen: penyisipan atau penghapusan elemen di tengah *Array* memerlukan pergeseran elemen, yang dapat memakan waktu pada *Stack* yang besar. *Linked list*: Kelebihan: 1. Ukuran dinamis: memungkinkan ukuran *Stack* untuk bertambah atau berkurang secara dinamis tanpa memerlukan pembuatan ulang atau alokasi memori baru. 2. Penyisipan dan penghapusan efisien: penyisipan dan penghapusan elemen di tengah *Stack* dapat dilakukan dengan efisien tanpa pergeseran elemen. 3. Alokasi memori dinamis: memungkinkan alokasi memori dinamis, yang dapat mengurangi fragmentasi memori dan memanfaatkan memori secara efisien. Kekurangan: 1. Akses tidak langsung: akses ke elemen tertentu memerlukan penelusuran dari awal hingga elemen yang diinginkan, sehingga memiliki kompleksitas waktu o(n). 2. Overhead tautan: memerlukan overhead tambahan untuk menyimpan tautan antar simpul, yang dapat meningkatkan penggunaan memori dan kompleksitas implementasi. 3. Keamanan memori: memerlukan manajemen memori dinamis yang baik untuk mencegah risiko keamanan memori, seperti referensi yang rusak. Pilihan antara *Array* dan *Linked list* tergantung pada kebutuhan spesifik aplikasi. Jika ukuran *Stack* statis dan akses elemen yang cepat diperlukan, *Array* mungkin menjadi pilihan yang lebih baik. Di sisi lain, jika ukuran *Stack* dinamis dan operasi penyisipan/penghapusan sering terjadi, *Linked list* dapat lebih efisien. Keduanya memiliki trade-off tertentu, dan pemilihan tergantung pada skenario penggunaan dan preferensi performa.

Hasil dan Evaluasi

evaluasi: Kecepatan akses: Pendekatan *Array*: lebih cepat karena dapat mengakses elemen langsung melalui indeks. Pendekatan *Linked list*: membutuhkan traversal untuk mengakses elemen, yang bisa memakan waktu lebih lama. Fleksibilitas ukuran: Pendekatan *Array*: ukuran tetap, sulit untuk diubah secara dinamis. Pendekatan *Linked list*: fleksibel dalam hal ukuran, dapat diubah sesuai kebutuhan. Pemakaian memori: Pendekatan *Array*: lebih efisien dalam hal pemakaian memori karena tidak ada overhead pointer. Pendekatan *Linked list*: memerlukan alokasi memori tambahan untuk setiap node. Performa operasi push dan pop: Pendekatan *Array*: operasi push dan pop cepat, tetapi mungkin

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

memerlukan pergeseran elemen. Pendekatan *Linked list*: operasi push dan pop biasanya lebih cepat karena tidak ada pergeseran elemen

SIMPULAN

Pilihan antara pendekatan *Array* atau pendekatan *Linked list* bergantung pada kebutuhan spesifik aplikasi dan preferensi desain anda. Jika kecepatan akses dan penggunaan memori yang efisien penting, pendekatan *Array* mungkin lebih tepat. Jika fleksibilitas ukuran dan operasi push/pop yang cepat merupakan prioritas, pendekatan *Linked list* mungkin merupakan pilihan yang lebih baik. Harap dicatat bahwa peringkat kinerja mungkin sangat bergantung pada skenario penggunaan spesifik dan ukuran data. Saat mengembangkan perangkat lunak, penting untuk mempertimbangkan trade-off antara kecepatan, penggunaan memori, dan fleksibilitas, tergantung pada kebutuhan proyek. Pilihan antara *Array* dan *Linked list* bergantung pada kebutuhan spesifik aplikasi anda. Jika anda memerlukan ukuran tumpukan statis dan akses elemen yang cepat, *Array* mungkin merupakan pilihan yang lebih baik. Di sisi lain, jika ukuran batch bersifat dinamis dan operasi penyisipan/penghapusan sering terjadi,*Linked list* akan lebih efisien. Pemahaman mendalam tentang pro dan kontra keduanya adalah kunci keputusan penerapan anda.

DAFTAR RUJUKAN

Ahmad, A. (2024). *Algoritma dan Pemrograman*. https://www.researchgate.net/publication/382560738

Alphacodingskills. (n.d.). *Data Structure - Circular Doubly Linked List*. Https://Www.Alphacodingskills.Com/Ds/Circular-Doubly-Linked-List.Php.

Ayoosh Sharma. (2021, June 9). *Circular Singly Linked List*. Https://Javadevjournal.Com/Data-Structure/Circular-Singly-Linked-List/.

Geeksforgeeks. (2024a, August 12). *Doubly Linked List Tutorial*. Https://Www.Geeksforgeeks.Org/Doubly-Linked-List/.

geeksforgeeks. (2024b, December 6). *What is Stack Data Structure? A Complete Tutorial*. https://Origin.Geeksforgeeks.Org/Introduction-to-Stack-Data-Structure-and-Algorithm-Tutorials/.

Hafiz, S., Ginting, N., Effendi, H., Kumar, S., Marsisno, W., Ria, Y., Sitanggang, U., Anwar, K., Putu, N., Santiari, L., Setyowibowo, S., Sigar, R., Atho'illah, I., Setyantoro, D., Nyoman, N., & Smrti, E. (2023). *Pengantar Struktur Data PT. MIFANDI MANDIRI DIGITAL*.

Indah Kusuma Astuti. (2019). *STRUKTUR DATA LINKED LIST*. https://doi.org/https://doi.org/10.31219/osf.io/8pj27

JUTECH
Journal Education and Technology
http://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/index
E - ISSN 2722-9297
APTIKOM

Joko Musridho, R. (2024). *Diktat STRUKTUR DATA*. https://repository.universitaspahlawan.ac.id/1887/1/%282024%29%20Musridho%20-%20Diktat%20Struktur%20Data.pdf

Mustakim, Dahlan Susilo, Mokhammad Syafaat, Amirul Mukminin, Deny Ariestiandy, Febrya Christin H. Buan, Edy Wihardjo, Luisa Sentia Paly, Erna Juniasti Malaikosa, & Nur Hayati. (2024). *STRUKTUR DATA DAN ALGORITMA* (Paput Tri Cahyono, Ed.). Yayasan Cendikia Mulia Mandiri.

Purwono, Alfian Ma Arif, & Iswanto. (2023). *Belajar Struktur Data dengan Python Penerbit UHB Press* (S. Kom. , M. K. Imam Ahmad Ashari, Ed.; edisi Pertama). UHB Press.

Putri, G. M., Pradja, K. A. Di, Azizi, M. B. M., Nurwahid, P., Perdana, A. S., & . M. (2024). Implementasi Stack dan Array pada Pengurutan Lagu dengan Metode Selection Sort. *Jurnal Teknologi Dan Sistem Informasi Bisnis*, *6*(2), 286–296. https://doi.org/10.47233/jteksis.v6i2.1192

Melati, Sari. (2024, February 9). *Perbedaan Array 1 Dimensi dan 2 Dimensi: Dari Mhandling Kebohongan Hingga Pemisahan Semesta Cendekia*. https://Tambahpinter.Com/Perbedaan-Array-1-Dimensi-Dan-2-Dimensi/.

Setiawan, A. (n.d.). *MODUL STRUKTUR DATA SPIRIT TOWAR|D THE FUTURE 2021*. Retrieved December 14, 2024, from https://www.agung73.com/wp-content/uploads/2021/02/Modul-Struktur-Data-R1.pdf

Sihombing, J., Ganesha, P., Gatot, J., No, S., & Bandung. (2020). *Penerapan Stack dan Queue Pada Array dan Linked List Dalam Java*.

Soffya Ranti. (2022). *Pengertian Stack dan Queue serta Contoh Penerapannya*. Kompas.Com. https://tekno.kompas.com/read/2022/12/01/02150047/pengertian-stack-dan-queue-serta-contoh-penerapannya?page=all#

Triase. (2020). *STRUKTUR DATA*. http://repository.uinsu.ac.id/9717/2/Diktat%20Struktur%20Data.pdf

Trivusi. (2022). *Struktur Data Stack: Pengertian, Karakteristik, dan Kegunaannya*. Trivusi.Web.Id.