Jurnal Riset Multidisiplin Edukasi
https://journal.hasbaedukasi.co.id/index.php/jurmie
Halaman : 147-164

SIMULATOR INTERAKTIF SINGLE LINKED LIST BERBASIS STEP EXECUTION UNTUK MEMPELAJARI MEKANISME PENUNJUKAN DAN PENGELOLAAN MEMORI

Edison Pardengganan Siahaan
Teknik Informatika – Fakultas Teknik Universitas Mpu Tantular
Email: edisonsiahaan@mputantular.ac.id

Informasi

Volume : 3
Nomor : 6
Bulan : Juni
Tahun : 2026
E-ISSN : 3062-9624

Abstract

The singly linked list data structure is a fundamental concept in computer science that is often difficult to understand due to dynamic memory allocation, pointers, and changes in node relationships. This study aims to design and develop an interactive desktop-based visualization software (Java Swing) that can graphically depict a singly linked list, provide step by step simulation for insert, delete, update, and traverse operations, and synchronize the visualization with highlighted line by line C code snippets. The development method follows the iterative Unified Process with four phases: Inception, Elaboration, Construction, and Transition. In the Elaboration phase, artifacts produced include fully dressed use cases, a use case diagram, system sequence diagrams (SSDs), a design class diagram, and a low fidelity wireframe user interface. Test results show that the application successfully executes all operation scenarios, including edge conditions (empty list, single node, invalid positions). The step simulation feature with commit/non commit mechanism and backward/forward navigation works as designed. The memory panel and code highlighting help clarify the relationship between C code and changes in the data structure. This application is suitable for use as an interactive learning medium in Data Structures courses, particularly on the topic of singly linked lists.

Keyword: *Singly linked list, data structure visualization, step by step simulation, memory management, Java Swing, computer science education, Unified Process.*

Abstrak

Struktur data singly linked list merupakan konsep fundamental dalam ilmu komputer yang seringkali sulit dipahami karena melibatkan alokasi memori dinamis, pointer, dan perubahan hubungan antar simpul. Penelitian ini bertujuan merancang dan membangun perangkat lunak visualisasi interaktif berbasis desktop (Java Swing) yang dapat menggambarkan singly linked list secara grafis, menyediakan simulasi langkah demi langkah untuk operasi insert, delete, update, dan penelusuran, serta menyinkronkan visualisasi dengan potongan kode C yang disorot baris per baris. Metode pengembangan menggunakan model iteratif unified process dengan empat fase: Inception, Elaboration, Construction, dan Transition. Pada fase Elaboration dihasilkan artefak berupa use case fully dressed, use case diagram, system sequence diagram (SSD), design class diagram dan Low Fidelity Wireframe User Interface. Hasil pengujian menunjukkan bahwa perangkat lunak berhasil menjalankan seluruh skenario operasi dengan benar, termasuk kondisi batas (list kosong, satu node, posisi tidak valid). Fitur simulasi langkah dengan mekanisme commit/non commit dan navigasi mundur/maju bekerja sesuai rancangan. Panel memori dan penyorotan kode membantu memperjelas hubungan antara kode C dan perubahan struktur data. Perangkat lunak ini layak digunakan sebagai media pembelajaran interaktif untuk mata kuliah Struktur Data, khususnya topik singly linked list.

Kata Kunci: *Singly linked list, visualisasi struktur data, simulasi step by step, manajemen memori, Java Swing, pendidikan komputer, Unified Process.*

A. PENDAHULUAN

Struktur data merupakan fondasi utama dalam rekayasa perangkat lunak dan ilmu komputer. Salah satu struktur data paling fundamental adalah *linked list*, khususnya *singly linked list*. Berbeda dengan *array* yang menggunakan alokasi memori statis dan berurutan, *singly linked list* mengandalkan alokasi memori dinamis di mana setiap elemen (simpul) terhubung melalui *pointer*. Konsep ini memungkinkan penyisipan dan penghapusan elemen secara efisien tanpa pergeseran data, namun sekaligus menjadi tantangan besar bagi mahasiswa pemula karena melibatkan abstraksi alamat memori, *pointer*, dan hubungan antar simpul (Cormen et al., 2022).

Berdasarkan pengalaman pengajaran mata kuliah Struktur Data, banyak mahasiswa mengalami kesulitan dalam:

- Representasi logis *linked list* versus implementasi fisik di memori.
- Mekanisme perubahan *pointer* saat menyisipkan atau menghapus simpul di posisi *head*, *tail*, maupun di antara.
- Hubungan antara kode program dalam bahasa C (*malloc, free,* dan *operasi* → *next*) dengan keadaan *linked list* yang sebenarnya.
- Konsep alamat memori dan bagaimana *pointer* menyimpan alamat tersebut.

Media pembelajaran konvensional seperti diagram statis di papan tulis atau slide presentasi tidak cukup untuk menunjukkan proses dinamis yang terjadi saat operasi *linked list* dieksekusi. Beberapa alat bantu visualisasi daring (misalnya oleh University of San Francisco (Galles, n.d.) dan VisuAlgo (Halim & Koh, 2011)) telah tersedia, namun umumnya masih terbatas pada animasi tanpa kode yang tersinkronisasi, tidak menyertakan representasi alamat memori, serta tidak mendukung navigasi langkah mundur/maju secara bebas.

Oleh karena itu, diperlukan sebuah perangkat lunak visualisasi interaktif yang dirancang khusus untuk membantu memahami *singly linked list* secara komprehensif. Perangkat lunak yang dikembangkan dalam penelitian ini tidak hanya menampilkan simpul dan *link* secara grafis, tetapi juga menyediakan simulasi **langkah demi langkah** (*step-by-step*) untuk setiap operasi (insert, delete, update, traverse), menyoroti baris kode C yang sedang dieksekusi,

serta merepresentasikan alamat memori virtual dengan sehingga pengguna dapat mengamati bagaimana *pointer* bekerja di tingkat memori.

Rumusan Masalah Berdasarkan latar belakang tersebut, rumusan masalah dalam penelitian ini adalah:

1. Bagaimana merancang dan membangun perangkat lunak desktop yang dapat memvisualisasikan struktur data *singly linked list* secara interaktif?
2. Bagaimana mengintegrasikan simulasi langkah demi langkah untuk operasi-operasi dasar *singly linked list* (insert head/tail/between, delete head/tail/between, update head/tail/between, dan penelusuran) dengan penyorotan kode C yang sinkron?
3. Bagaimana merepresentasikan alamat memori virtual pada setiap simpul sehingga konsep *pointer* dan alokasi memori dinamis lebih mudah dipahami?
4. Bagaimana menguji kebenaran fungsionalitas perangkat lunak melalui pengujian *black-box*?

Tujuan Penelitian

Tujuan dari penelitian ini adalah:

1. Menghasilkan perangkat lunak visualisasi *singly linked list* berbasis *desktop* (Java Swing) dengan fitur simulasi langkah, visualisasi grafis simpul, dan panel memori.
2. Menyediakan sinkronisasi antara tampilan grafis dan potongan kode C untuk setiap operasi, termasuk penyorotan baris kode yang sedang dieksekusi.
3. Melakukan uji coba fungsionalitas (*black-box testing*) untuk memastikan perangkat lunak bebas dari kesalahan logika pada semua skenario operasi.

Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini adalah:

- **Bagi dosen/pengajar**: Tersedianya alat bantu mengajar yang interaktif untuk menjelaskan materi *linked list* secara lebih konkret dan menarik.
- **Bagi mahasiswa/pembelajar**: Memudahkan dalam memahami alur algoritma, perubahan *pointer*, serta hubungan antara kode dan memori.
- **Bagi pengembang perangkat lunak pendidikan**: Menjadi contoh implementasi visualisasi struktur data yang dapat dikembangkan lebih lanjut untuk topik lain seperti *doubly linked list*, *stack*, *queue*, atau *tree*.

Batasan Penelitian

Agar penelitian lebih terfokus, diberikan batasan-batasan sebagai berikut:

- Perangkat Lunak hanya memvisualisasikan ***singly linked* list** dengan penelusuran dari *head* (bukan dari *tail*).
- Operasi yang disediakan meliputi: *insert* (head, tail, between), *delete* (head, tail, between), *update* (head, tail, between), dan penelusuran.
- Kode yang ditampilkan dan disinkronkan adalah kode dalam bahasa C (sesuai dengan modul ajar yang digunakan).
- Perangkat Lunak dikembangkan menggunakan **Java Swing** dan dijalankan pada lingkungan *desktop* (tidak berbasis web).
- Pengujian dilakukan secara fungsional (*black-box*)

Tinjauan Pustaka

Konsep Singly Linked List

Singly linked list adalah struktur data linear di mana setiap elemen (disebut simpul atau *node*) terdiri dari dua bagian utama: **data** dan **pointer** (biasanya bernama next) yang menunjuk ke simpul berikutnya. Simpul pertama disebut *head*, simpul terakhir disebut *tail*; *tail* memiliki pointer next yang bernilai NULL sebagai penanda akhir *list* (Cormen et al., 2022). Untuk *singly linked list* yang mengizinkan penelusuran satu arah (dari *head* ke *tail*) maka karakteristik ini membuat operasi penyisipan dan penghapusan di bagian *head* sangat efisien (O(1)), sedangkan operasi di posisi tertentu memerlukan penelusuran (O(n)). Implementasi dalam bahasa C umumnya akan menggunakan struct dengan *field* data dan next, serta fungsi-fungsi seperti penciptaan simpul, penyisipan simpul di bagian head, tail dan diantara head-tail, penghapusan simpul di bagian head, bagian tail dan diantara head-tail, pembaruan data dan penelusuran.

Manajemen Memori dalam *Linked List*

Salah satu konsep paling abstrak dalam *linked list* adalah **manajemen memori dinamis**. Dalam bahasa C, simpul dialokasikan menggunakan malloc yang mengembalikan alamat memori di heap. Setiap simpul menempati blok memori tertentu, dan pointer next menyimpan alamat simpul berikutnya. Pemahaman tentang alamat memori sangat penting untuk menjelaskan bagaimana perubahan pointer dapat menyambung atau memutus hubungan antar simpul (Kernighan & Ritchie, 1988).

Pada perangkat lunak visualisasi yang dikembangkan, alamat memori dari simpul akan disimulasikan secara dinamis sehingga setiap simpul memiliki alamat yang unik. Hal ini dilakukan untuk tujuan pembelajaran, dan memudahkan pemahaman hubungan antara alamat yang disimpan oleh pointer dan keberadaan simpul.

Media Pembelajaran Visualisasi Struktur Data

Penelitian tentang visualisasi struktur data untuk pendidikan telah banyak dilakukan. Beberapa alat yang terkenal antara lain:

- **Data Structure Visualizations** (University of San Francisco; Galles, n.d.):Menyediakan animasi untuk berbagai struktur data, termasuk *linked list*. Namun alat ini tidak menyediakan sinkronisasi dengan kode program secara langsung, dan tidak mendukung navigasi mundur/maju per langkah secara granular (hanya animasi kontinu).
- **VisuAlgo** (National University of Singapore; Halim & Koh, 2011): Menampilkan visualisasi interaktif dengan penjelasan teks, tetapi juga tidak menyertakan kode C yang tersinkronisasi per baris.
- **Python Tutor** (Guo, 2013): Memvisualisasikan eksekusi kode Python, Java, JavaScript, C, C++ dengan *step-by-step* dan menunjukkan *stack* dan *heap*, tetapi tidak spesifik untuk *linked list* dan tidak dirancang untuk menyoroti operasi *linked list* secara grafis.

Perangkat lunak yang dikembangkan dalam penelitian ini melampaui alat-alat yang ada dengan mengintegrasikan tiga komponen sekaligus:

- Visualisasi grafis simpul dan *link* (dengan warna berbeda untuk node normal, aktif, baru, dan akan dihapus).
- Panel memori yang menunjukkan alamat, data, dan pointer next setiap simpul.
- Panel kode C yang menyoroti baris yang sedang dieksekusi, serta penjelasan teks untuk setiap langkah.

Teori Step-by-Step Simulation dan Code Highlighting

Prinsip pedagogi yang mendasari desain perangkat lunak ini adalah **pembelajaran bertahap** dan **pemetaan langsung antara kode dan eksekusi**. Menurut Mayer (2014), dalam *Cognitive Theory of Multimedia Learning*, pembelajar lebih memahami materi ketika informasi verbal (teks/kode) dan visual (diagram) disajikan secara simultan dan terintegrasi. Simulasi langkah per langkah memungkinkan pembelajar untuk mengontrol kecepatan belajar, mengulang bagian yang sulit, serta melihat hubungan sebab-akibat dari setiap instruksi kode.

Code highlighting (penyorotan baris) secara *real-time* telah terbukti efektif untuk meningkatkan *code comprehension*, terutama pada pemrogram pemula (Kelleher & Pausch, 2005). Dengan menyoroti baris yang sedang dieksekusi di panel kode, pembelajar dapat

secara langsung melihat bahwa baris *simpulBaru->next = list->head* mengakibatkan perubahan grafis di kanvas, misalnya munculnya panah dari simpul baru ke head lama.

Penelitian Terkait

Beberapa penelitian sejenis yang relevan dengan topik ini antara lain ditunjukkan pada Tabel 1.

Tabel 1. Perbandingan dengan penelitian terkait

Peneliti (Tahun)	Fokus	Kelebihan	Kekurangan
Smith (2019)	*Interactive Linked List Visualization for CS1*	Animasi warna untuk perubahan pointer	Tidak ada sinkronisasi kode, navigasi terbatas
Chen dkk. (2021)	*A Web-Based Tool for Visualizing Dynamic Data Structures*	Akses via web, mendukung beberapa struktur data	Tidak menyediakan model memori; hanya animasi global
Pratama (2022)	Pengembangan media pembelajaran *linked list* berbasis Android	Mobile, mudah digunakan	Fitur *step-by-step* terbatas, tidak ada representasi alamat
Penelitian ini (2025)	Perangkat lunak desktop dengan simulasi *step*, kode C sinkron, model memori statis, navigasi maju/mundur	Integrasi tiga panel, penyorotan baris, alamat virtual 8-byte, dukungan *update data*	Masih terbatas pada *singly linked list*

Dari tinjauan pustaka di atas, dapat disimpulkan bahwa belum ada perangkat lunak visualisasi *singly linked list* yang secara lengkap menggabungkan simulasi struktur data singly linked list dalam bentuk visual yang disinkronisasi dengan perubahan highlight di kode C.

Penerapan dan Penggunaan Unified Process (UP) secara Ringan dan Agile Menurut Craig Larman

Penelitian ini akan menggunakan Unified Process secara ringan dan agile seperti yang disampaikan oleh Craig Larman (2005). Unified Process adalah kerangka kerja pengembangan perangkat lunak berorientasi objek yang iteratif dan inkremental. Unified Process memiliki empat fase utama:

- **Inception**: Menentukan visi, ruang lingkup, mengidentifikasi kebutuhan, dan membuat *business case*.

- **Elaboration**: Membangun arsitektur inti, mengimplementasikan *use case* yang signifikan secara arsitektur, membuat analisis dan desain rinci (termasuk *use case diagram*, *system sequence diagram*, *activity diagram*, *class diagram*, dan *domain model*).
- **Construction**: Membangun fitur sisanya secara bertahap melalui beberapa iterasi.
- **Transition**: Menguji, menerapkan, dan menyerahkan produk ke pengguna.

Pada fase Elaboration, artefak penting seperti *use case* yang *fully dressed*, *system sequence diagram* (SSD), *activity diagram*, dan *class diagram* mulai dikembangkan untuk memandu implementasi (Larman, 2005).

Teknologi yang Digunakan

Perangkat lunak visualisasi ini dibangun menggunakan:

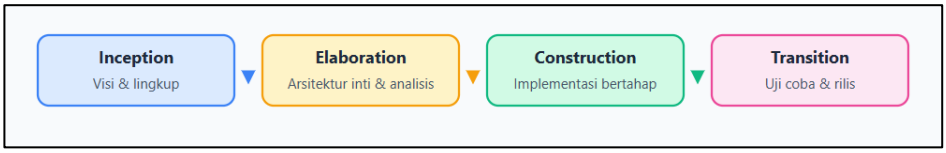
- **Bahasa pemrograman Java** (versi 8 atau lebih tinggi) dengan pustaka **Swing** untuk antarmuka pengguna grafis.
- ***Custom painting*** (Graphics2D) untuk menggambar simpul, panah, dan teks pada kanvas.
- ***Model-View pattern*** untuk memisahkan logika data (LinkedListModel) dari tampilan (VizCanvas).
- **Struktur data** SimulationStep untuk menyimpan setiap langkah simulasi (pesan, indeks node yang disorot, baris kode, perubahan model, dll.).
- **Mekanisme *Highlighter* dari** JTextArea untuk menyoroti baris kode C secara dinamis.

Kode sumber perangkat lunak (diberikan dalam LinkedListVisualizer.java) mengimplementasikan semua fitur yang disebutkan, dan menjadi artefak utama dari penelitian ini.

B. METODE PENELITIAN

Jenis Penelitian

Penelitian ini termasuk dalam kategori **penelitian pengembangan** (*development research*) karena bertujuan menghasilkan produk berupa perangkat lunak perangkat lunak visualisasi *singly linked list* (Borg & Gall, 2007). Pendekatan yang digunakan adalah **Unified Process (UP)** versi Craig Larman (Larman, 2005) dengan empat fase iteratif seperti ditunjukkan pada Gambar 1.



Gambar 1. Fase-fase Unified Process (UP)

Fase Inception

Fase *inception* merupakan tahap awal dalam siklus pengembangan yang bertujuan untuk menetapkan fondasi proyek secara menyeluruh, mencakup visi produk, ruang lingkup sistem, identifikasi risiko, serta pendefinisian kebutuhan fungsional awal.

Visi Produk. Produk yang dikembangkan adalah perangkat lunak visualisasi *singly linked list* berbasis antarmuka grafis yang menyediakan mekanisme simulasi operasi secara *step-by-step* dengan sinkronisasi terhadap kode pemrograman bahasa C. Perangkat lunak ini dirancang untuk mendukung pemahaman konseptual struktur data secara visual dan interaktif, khususnya bagi pengguna yang sedang mempelajari implementasi *linked list* pada level pemrograman sistem.

Ruang Lingkup Sistem. Sistem dibatasi pada implementasi dan visualisasi struktur data *singly linked list*. Operasi yang didukung meliputi empat fungsi utama, yaitu penyisipan node (*insert*), penghapusan node (*delete*), pembaruan nilai node (*update*), dan penelusuran seluruh node (*traverse*). Pembatasan ruang lingkup pada *singly linked list* dilakukan secara deliberatif untuk menjaga fokus pengembangan dan menghindari kompleksitas yang tidak relevan dengan tujuan pembelajaran yang disasar.

Identifikasi Risiko Utama. Berdasarkan analisis awal, terdapat tiga risiko utama yang diidentifikasi pada fase ini. Pertama, kompleksitas sinkronisasi antara representasi kode C, visualisasi grafis node, dan mekanisme navigasi langkah-per-langkah yang harus berjalan secara konsisten. Kedua, tantangan dalam merepresentasikan model memori virtual secara akurat sehingga dapat mencerminkan perilaku alokasi dan dealokasi memori yang terjadi secara nyata pada implementasi C. Ketiga, penanganan kondisi batas (*edge cases*) yang mencakup skenario list kosong, list dengan satu node, serta masukan posisi yang tidak valid, yang berpotensi menyebabkan kegagalan logika apabila tidak ditangani secara eksplisit.

Identifikasi *Use Case*. Pada fase ini ditetapkan tujuh *use case* awal yang merepresentasikan keseluruhan fungsionalitas sistem, sebagaimana ditampilkan pada Tabel 2. dan diagram Use Case pada gambar 2.

Tabel 2. Daftar *Use Case* Awal Sistem

Kode UC	Nama *Use Case*	Deskripsi Singkat
UC-01	*Insert Node*	Pengguna menyisipkan node baru pada posisi tertentu dalam list
UC-02	*Delete Node*	Pengguna menghapus node berdasarkan posisi atau nilai tertentu

UC-03	*Update Node*	Pengguna memperbarui nilai data pada node yang dipilih
UC-04	*Traverse List*	Sistem menelusuri seluruh node dari kepala hingga akhir list
UC-05	*Step-by-Step Simulation*	Pengguna menjalankan simulasi operasi secara bertahap dengan kontrol manual
UC-06	*View Memory Panel*	Sistem menampilkan representasi virtual memori untuk setiap node
UC-07	*Reset List*	Pengguna mengatur ulang list ke kondisi awal kosong

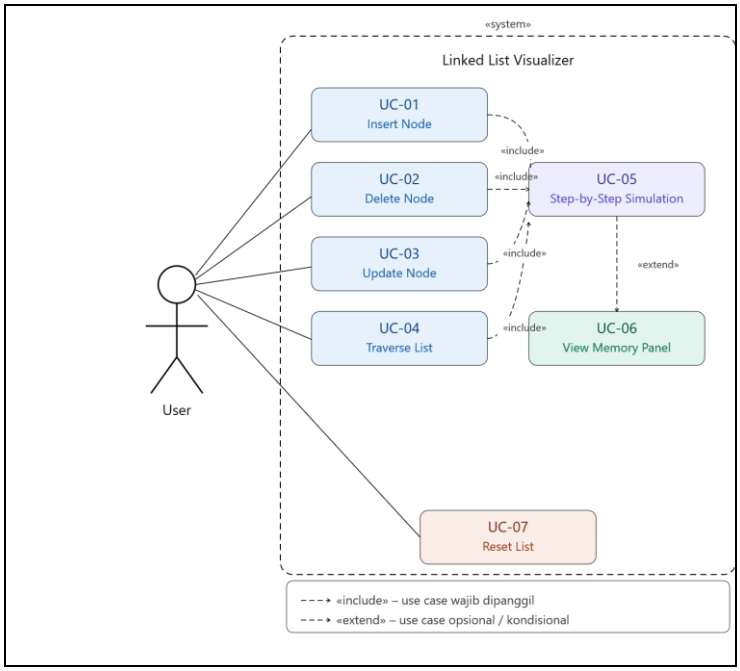
Fase Elaboration

Fase *elaboration* merupakan fase paling kritis dalam siklus pengembangan iteratif karena bertujuan untuk membangun *architectural baseline* yang stabil sekaligus mereduksi risiko-risiko teknis yang telah diidentifikasi pada fase sebelumnya (Larman, 2005). Pada fase ini, kebutuhan sistem dianalisis secara lebih mendalam dan dituangkan ke dalam berbagai artefak perancangan yang menjadi acuan implementasi.

Aktivitas yang dilakukan pada fase ini meliputi: (1) pembuatan *use case diagram* yang menggambarkan hubungan antara aktor dan seluruh *use case* sistem; (2) analisis *use case* secara rinci (3) pembuatan *System Sequence Diagram* (SSD); (4) pembuatan *design class diagram*; serta (5) pembuatan rancangan antar muka (*user interface*).

Use Case Diagram dan Spesifikasi Use Case Fully Dressed – *Insert Node*

Gambar 2 menunjukkan diagram use case yang menggambarkan semua fungsionalitas utama yang dapat dilakukan oleh aktor *User*.



Gambar 3. Use Case Diagram Perangkat lunak Visualisasi Singly Linked List

Dari ketujuh *use case* yang telah diidentifikasi pada fase *inception*, UC-01 *Insert Node*

adalah salah satu use case yang akan ditunjukkan bentuk deskripsinya dalam fully dressed. UC-01 adalah salah satu use case yang dapat menggambarkan dan merepresentasikan kompleksitas arsitektur sistem yang sedang direkayasa. Use case ini akan mencakup mekanisme simulasi bertahap, sinkronisasi kode C, pembaruan visualisasi grafis, dan pengelolaan panel memori secara bersamaan. Spesifikasi deskripsi lengkap *use case* ini disajikan pada Tabel 3.

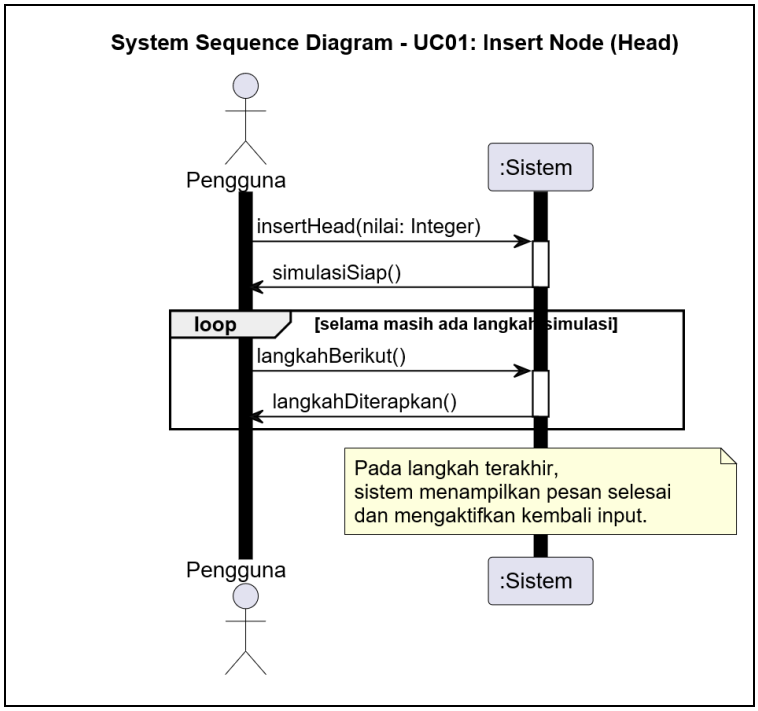
Tabel 3. Spesifikasi Use Case Fully Dressed UC-01 *Insert Node*

Elemen	Deskripsi
Nama *Use Case*	*Insert Node*
Kode	UC-01
Aktor	*User*
Kepentingan *Stakeholder*	*User* ingin menambahkan node baru di awal *linked list* dan mengamati simulasi perubahan struktur memori secara bertahap dan terurut
Prakondisi (*Preconditions*): **1. Perangkat lunak telah berjalan dan antarmuka siap menerima masukan.** **2. *Linked list* berada dalam kondisi kosong atau telah memuat sejumlah node.** **Pascakondisi (*Postconditions*):** **1. Node baru dengan nilai yang ditentukan menjadi *head* baru dari *linked list*.** **2. Ukuran *linked list* bertambah satu satuan.** **3. Simulasi langkah-langkah eksekusi kode siap dinavigasi oleh pengguna.** **Skenario Utama (Main Success Scenario):** **1. *User* memilih operasi "*Insert*" dan suboperasi "*Head*" pada antarmuka.** **2. *User* memasukkan nilai bilangan bulat (contoh: 42) pada *field* masukan nilai.** **3. *User* menekan tombol "Jalankan" untuk memulai simulasi.** **4. Sistem membangkitkan daftar langkah simulasi (*SimulationStep*) yang sesuai dengan operasi *insert head*.** **5. Sistem menonaktifkan *field* masukan dan mengaktifkan tombol navigasi "Berikut".** **6. *User* menekan tombol "Berikut" secara berulang untuk menelusuri setiap langkah simulasi, yang meliputi: (a) pembuatan node baru; (b) penghubungan node baru ke *head* lama melalui atribut *next*; (c) penetapan *head* = node baru; (d) penambahan ukuran *list* sebesar satu; dan (e) pembaruan referensi *tail* apabila *list* sebelumnya dalam kondisi kosong.** **7. Pada langkah terakhir, sistem menampilkan notifikasi penyelesaian dan mengaktifkan kembali *field* masukan.** **8. Visualisasi *linked list* dan panel memori virtual diperbarui secara sinkron untuk mencerminkan kondisi terkini.** **Alur Alternatif (*Alternative Flows*):** **• *List* kosong: Setelah eksekusi *insert head*, referensi *tail* juga diarahkan ke node baru yang sekaligus menjadi satu-satunya node dalam *list*.** **• Nilai masukan tidak valid: Sistem menampilkan pesan *error* pada antarmuka dan simulasi tidak dijalankan hingga masukan yang valid diberikan.** **• *User* menekan "Sebelum": Sistem membatalkan langkah terakhir dan memulihkan *state* visualisasi ke kondisi langkah sebelumnya (*undo step*).** **Aturan Bisnis (*Business Rules*):** **• Setiap node memiliki identifikasi unik (*id*) yang menentukan alamat memori virtualnya, dihitung menggunakan formula: alamat = 0x2000 + id × 8.** **• Simulasi hanya dapat dijalankan apabila operasi yang diminta memenuhi validasi masukan yang telah ditetapkan.**	

System Sequence Diagram (SSD)

System Sequence Diagram (SSD) adalah diagram yang menggambarkan interaksi

antara aktor (dalam hal ini *User*) dan sistem sebagai sebuah *black box* untuk satu skenario use case tertentu. SSD fokus pada **pesan sistem** (*system events*) yang dikirim oleh aktor ke sistem, serta **respons sistem** (opsional) terhadap pesan tersebut. SSD tidak menjelaskan logika internal sistem, melainkan hanya operasi sistem apa yang dipanggil, parameter apa yang disertakan, dan urutan temporalnya (Larman, 2005,). Gambar 4 menunjukkan salah satu contoh SSD yang digunakan pada penelitian yaitu SSD dari operasi insert head.



Gambar 4. SSD dari operasi Insert Head

Berdasarkan SSD di atas, dapat diidentifikasi dua buah operasi sistem utama:

1. insertHead(nilai: Integer)

   Operasi ini dipanggil pertama kali oleh pengguna setelah memasukkan nilai (misal 42) dan menekan tombol "Jalankan". Sistem akan membangkitkan daftar langkah simulasi (*SimulationStep*) untuk operasi *insert head*.

2. langkahBerikut()

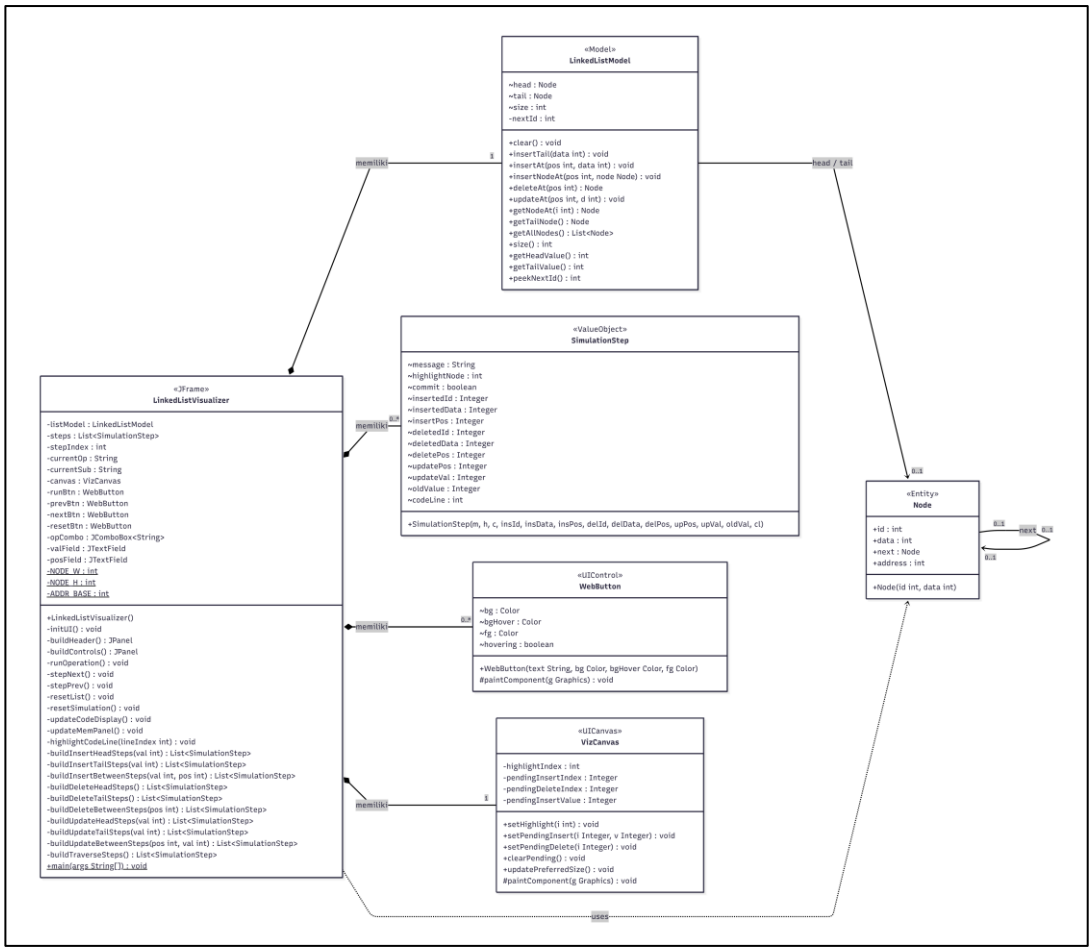
   Operasi ini dipanggil secara berulang oleh pengguna setiap kali menekan tombol "Berikut". Setiap pemanggilan menyebabkan sistem memajukan satu langkah simulasi, memperbarui tampilan visual linked list dan panel memori, lalu mengembalikan umpan balik **langkahDiterapkan()**. Sifat perulangan digambarkan dengan bingkai loop. Operasi langkahBerikut() ini sangat penting untuk memenuhi kebutuhan pengguna dalam mengamati perubahan status memori secara bertahap.

   Selain itu, SSD menunjukkan adanya **umpan balik awal** (simulasiSiap()) yang

memberitahu pengguna bahwa simulasi telah siap untuk dinavigasi. Pada langkah terakhir, sistem memberikan pesan penyelesaian dan mengaktifkan kembali masukan pengguna (sesuai dengan *postcondition* dari *use case*).

Design Class Diagram

Class Diagram pada Gambar 5 menggambarkan struktur statis perangkat lunak visualisasi linked list. Diagram ini memodelkan kelas-kelas utama serta relasi antar kelas yang digunakan untuk mendukung simulasi operasi linked list secara langkah demi langkah.



Gambar 5. Design Class Diagram perangkat lunak visualisasi linked list

Terdapat enam kelas utama:

1. **LinkedListVisualizer** – kelas utama yang bertindak sebagai antarmuka pengguna (JFrame).

   Kelas ini memiliki komposisi terhadap LinkedListModel, VizCanvas, WebButton, dan daftar SimulationStep. Kelas ini mengelola seluruh alur simulasi, termasuk pembangkitan langkah-langkah operasi (insert, delete, update, traversal) dan navigasi antar langkah.
2. **LinkedListModel** – kelas model yang merepresentasikan struktur data linked list.

Kelas ini memiliki atribut head, tail, dan size, serta menyediakan metode untuk manipulasi list (insert, delete, update, get). Relasi asosiasi dengan Node menunjukkan bahwa model memiliki referensi ke simpul pertama dan terakhir.

3. **Node** – kelas entitas yang merepresentasikan satu simpul dalam linked list. Setiap node memiliki id, data, address (simulasi alamat memori), serta referensi next ke node berikutnya. Relasi asosiasi refleksif next dengan kardinalitas 0..4 menunjukkan bahwa setiap node hanya terhubung ke paling banyak satu node berikutnya dalam rantai.
4. **SimulationStep** – kelas value object yang merekam satu langkah simulasi. Atribut seperti highlightNode, insertedData, deletedId, codeLine, dll., menyimpan informasi perubahan status pada setiap langkah. Objek SimulationStep digunakan oleh visualizer untuk menampilkan animasi dan penjelasan teks secara bertahap.
5. **VizCanvas** – kelas turunan dari JPanel yang bertanggung jawab untuk menggambar visualisasi linked list. Kelas ini menangani penyorotan node, indikasi pending insert/delete, dan pengaturan ulang ukuran kanvas.
6. **WebButton** – kelas kustom untuk tombol dengan efek hover. Digunakan pada panel kontrol untuk navigasi simulasi (run, previous, next, reset).

Relasi antar kelas:

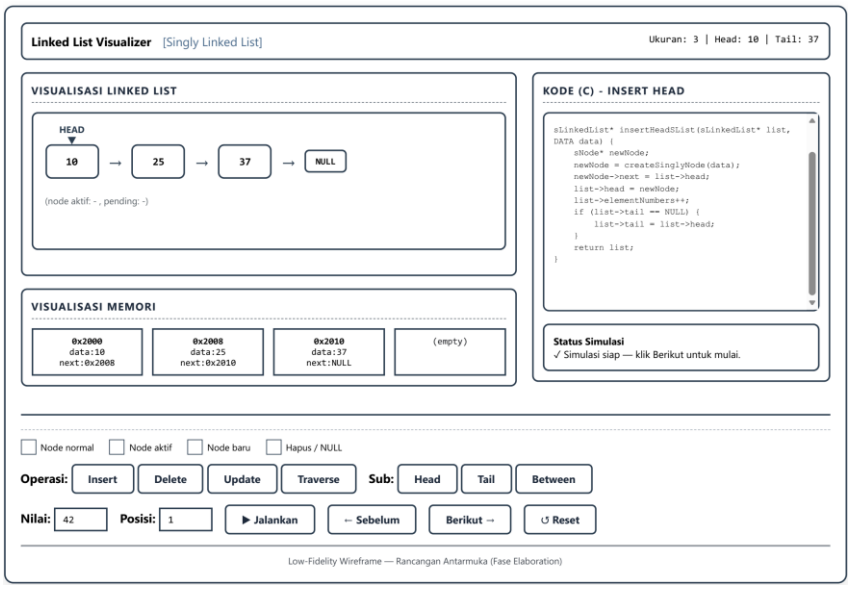
1. **Komposisi** (*--): LinkedListVisualizer memiliki tepat satu LinkedListModel, tepat satu VizCanvas, dan nol atau lebih SimulationStep serta WebButton. Ini menunjukkan bahwa siklus hidup objek-objek tersebut bergantung pada visualizer.
2. **Asosiasi** (-->): LinkedListModel terhubung ke 0 atau 1 Node sebagai head dan tail; Node memiliki asosiasi refleksif next dengan kardinalitas 0..1 (karena linked list dapat kosong atau memiliki banyak node, namun setiap node hanya memiliki satu next).
3. **Dependensi** (..>): LinkedListVisualizer menggunakan Node sebagai tipe parameter atau tipe lokal, namun tidak memiliki kepemilikan penuh.

Secara keseluruhan, class diagram ini menunjukkan arsitektur Model-View-Controller (dengan LinkedListModel sebagai model, VizCanvas sebagai view, dan LinkedList Visualizer sebagai controller) yang mendukung simulasi interaktif operasi linked list. Diagram ini menjadi landasan untuk implementasi sistem visualisasi yang diusulkan.

Rancangan Antar Muka (User Interface) Perangkat Lunak Visualisasi Linked List

Antarmuka perangkat lunak dirancang dengan prinsip *clean layout* dan *responsif*. Terdapat tiga area utama yaitu **Panel Kiri (Visualisasi), Panel Kanan (Kode & Status),** dan

Panel Bawah (Kontrol), seperti terlihat pada Gambar 6.



Gambar 6. Low Fidelity Wireframe Antar Muka Visualisasi Linked List

Fase Construction

Pada fase ini, implementasi kode dilakukan berdasarkan desain class diagram dan use case yang telah dirinci. Pengembangan dilakukan secara iteratif dengan prioritas:

- **Iterasi 1**: Implementasi model (LinkedListModel, Node) dan operasi dasar tanpa GUI.
- **Iterasi 2**: Implementasi GUI utama (LinkedListVisualizer, VizCanvas) dan visualisasi statis.
- **Iterasi 3**: Implementasi simulasi langkah (SimulationStep, navigasi maju/mundur).
- **Iterasi 4**: Integrasi kode C, panel memori, legenda, dan penyempurnaan antarmuka.

Hasil dari fase Construction adalah kode sumber LinkedListVisualizer.java yang siap untuk di test pada fase transition.

Fase Transition

Fase ini mencakup pengujian fungsionalitas, pembuatan dokumentasi pengguna, serta persiapan rilis. Salah satu aktivitas utama adalah **pengujian *black-box*** (*black-box testing*). Pengujian *black-box* adalah teknik pengujian perangkat lunak yang berfokus pada **spesifikasi fungsional** tanpa melihat struktur internal kode. Penguji hanya memberikan masukan dan mengamati keluaran yang dihasilkan oleh sistem, lalu membandingkannya dengan hasil yang diharapkan berdasarkan *use case* dan kebutuhan (Pressman, 2014). Tujuan pengujian ini adalah untuk menemukan kesalahan pada antarmuka, fungsi yang hilang, atau perilaku sistem yang tidak sesuai dengan spesifikasi.

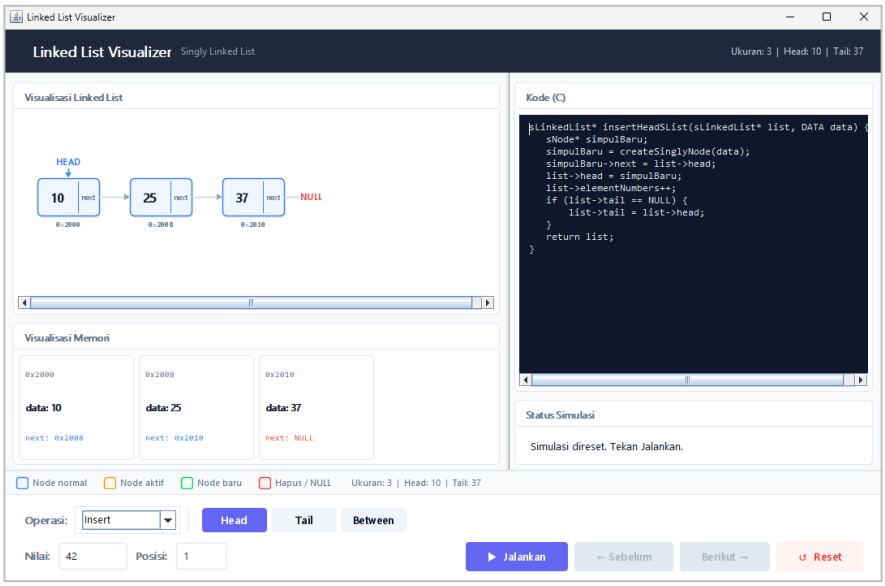
Pada penelitian ini, pengujian dilakukan dengan skenario seperti yang tercantum pada

Tabel 4 (hasil pengujian akan dibahas di bagian 4 tulisan ilmiah ini).

C. HASIL DAN PEMBAHASAN

Implementasi Antarmuka Perangkat lunak

Perangkat lunak visualisasi *singly linked list* berhasil diimplementasikan sesuai dengan rancangan pada Bagian 3. Gambar 7 menunjukkan tampilan utama perangkat lunak saat pertama kali dijalankan dengan tiga node default (10, 25, 37).



Gambar 7. Tampilan utama perangkat lunak dengan list default [10, 25, 37]

Antarmuka menggunakan skema latar belakang terang, node berwarna biru muda, dan tombol interaktif. Fitur legenda warna (normal, aktif, baru, hapus) membantu pengguna memahami status setiap node.

Hasil Pengujian Fungsionalitas

Pengujian *black-box* terhadap 16 skenario (Tabel 4) menunjukkan bahwa seluruh skenario berhasil dilalui. Tabel 5 merangkum hasil untuk beberapa skenario kritis.

Tabel 4. Skenario pengujian fungsionalitas

Operasi	Skenario	Hasil yang diharapkan
Insert Head	List kosong, nilai 10	Head = tail = node(10)
Insert Head	List [20,30], nilai 5	List menjadi [5,20,30]
Insert Tail	List kosong, nilai 10	Head = tail = node(10)
Insert Tail	List [20], nilai 30	List menjadi [20,30]
Insert Between	List [20,30], posisi 2, nilai 25	List menjadi [20,25,30]
Insert Between	Posisi <1 atau >size+1	Tidak ada perubahan, pesan error
Delete Head	List [10]	List kosong
Delete Head	List [10,20,30]	List menjadi [20,30]
Delete Tail	List [10]	List kosong
Delete Tail	List [10,20,30]	List menjadi [10,20]
Delete Between	List [10,20,30], posisi 2	List menjadi [10,30]
Update Head	List [10,20] → nilai 99	List menjadi [99,20]

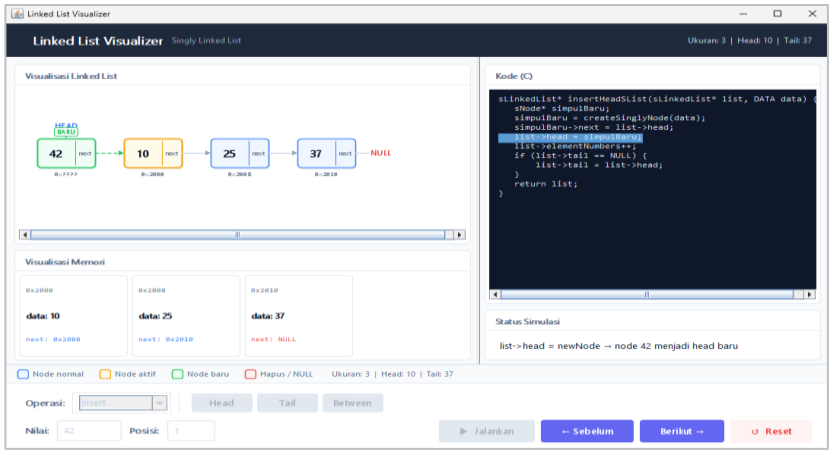
Update Tail	List [10,20] → nilai 99	List menjadi [10,99]
Update Between	List [10,20,30], posisi 2 → 99	List menjadi [10,99,30]
Penelusuran/ Traverse	List berisi 3 node	Menampilkan pesan (tidak mengubah model)

Tabel 5. Hasil pengujian fungsionalitas untuk skenario kritis

Operasi	Skenario	Hasil yang diamati	Status
Insert Head	List kosong, nilai 10	Kanvas menampilkan satu node (10) dengan HEAD dan tail menunjuk node tersebut; alamat 0x2000; next NULL.	✓
Insert Head	List [20,30], nilai 5	Node 5 muncul di paling kiri, panah dari 5 ke 20, head menunjuk 5, tail tetap di 30. Alamat node baru 0x2008.	✓
Insert Between	List [20,30], posisi 2, nilai 25	Node 25 muncul di antara 20 dan 30; panah 20→25→30; *elementNumbers* menjadi 3.	✓
Delete Head	List [10]	List kosong; kanvas menampilkan teks "List kosong (NULL)"; head dan tail NULL; panel memori kosong.	✓
Delete Between	List [10,20,30], posisi 2	Node 20 dihapus; panah langsung dari 10 ke 30; *elementNumbers* menjadi 2. Alamat node 20 tidak lagi ditampilkan.	✓
Update Between	List [10,20,30], posisi 2 → 99	Data node kedua berubah dari 20 menjadi 99 pada kanvas dan panel memori.	✓
Penelusuran/ Traverse	List [5,10,15]	Status menampilkan pesan berturut-turut: "Cetak node 1: 5", "cur = cur->next → node 2", dst. Tidak ada perubahan pada model.	✓

Sinkronisasi Kode dan Penyorotan Baris

Panel kode menampilkan potongan kode C yang sesuai dengan operasi yang dipilih. Setiap kali pengguna menekan "Berikut", baris kode yang sedang dieksekusi disorot dengan latar biru. Pengguna dapat secara langsung memetakan instruksi kode seperti simpulBaru->next = list->head ke perubahan visual (munculnya panah). Keterbatasan yang ditemukan adalah bahwa kode yang ditampilkan adalah *string literal* statis, bukan kode yang benar-benar dieksekusi oleh interpreter C. Namun untuk tujuan demonstrasi algoritma, hal ini sudah memadai karena fokusnya pada logika, bukan pada kompilasi. Gambar 8. Menunjukkan contoh tampilan sinkronisasi kode dengan penyorotan baris kode yang sedang aktif.



Gambar 8. Contoh Tampilan Penyorotan Kode Program yang sedang aktif

D. KESIMPULAN

Berdasarkan hasil penelitian dan pembahasan, dapat ditarik kesimpulan sebagai berikut:

1. **Perangkat lunak visualisasi *singly linked list* berhasil dirancang dan diimplementasikan** menggunakan Java Swing dengan antarmuka yang intuitif. Perangkat lunak mendukung operasi dasar (insert, delete, update, traversal) serta menyediakan simulasi langkah demi langkah, visualisasi grafis node dan panah, panel memori, serta sinkronisasi kode C.
2. **Simulasi langkah dengan mekanisme *commit/non-commit* dan navigasi mundur/maju** telah diimplementasikan dengan benar. Setiap langkah dapat diulang atau dibatalkan, membantu pemahaman perubahan pointer.
3. **Penyorotan baris kode C secara sinkron** dengan setiap langkah simulasi berfungsi dengan baik, menghubungkan instruksi kode dengan perubahan visual.
4. **Pengujian fungsionalitas (*black-box*)** menunjukkan bahwa perangkat lunak bebas dari kesalahan logika untuk semua skenario yang diuji, termasuk kondisi batas (list kosong, satu node, posisi tidak valid).

Dengan demikian, perangkat lunak yang dikembangkan layak digunakan sebagai media pembelajaran interaktif untuk mata kuliah Struktur Data, khususnya pada topik *singly linked list*.

E. DAFTAR PUSTAKA

Borg, W. R., & Gall, M. D. (2007). Educational research: An introduction (8th ed.). Pearson.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms (4th ed.). MIT Press.

Galles, D. (n.d.). Data structure visualizations. University of San Francisco. https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Guo, P. J. (2013). Online Python Tutor: Embeddable web-based program visualization for novices. Proceedings of the 2013 ACM Technical Symposium on Computer Science Education, 579–584. https://doi.org/10.1145/2445196.2445368

Halim, S., & Koh, F. (2011). VisuAlgo: Visualising data structures and algorithms through animation. Proceedings of the 3rd National Conference on Computer Science Education. National University of Singapore. https://visualgo.net

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, 37(2), 83–137. https://doi.org/10.1145/1089733.1089734

Kernighan, B. W., & Ritchie, D. M. (1988). The C programming language (2nd ed.). Prentice Hall.

Larman, C. (2005). Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development (3rd ed.). Prentice Hall.

Mayer, R. E. (Ed.). (2014). The Cambridge handbook of multimedia learning (2nd ed.). Cambridge University Press. https://doi.org/10.1017/CBO9781139547369

Pressman, R. S. (2014). Software engineering: A practitioner's approach (8th ed.). McGraw-Hill.