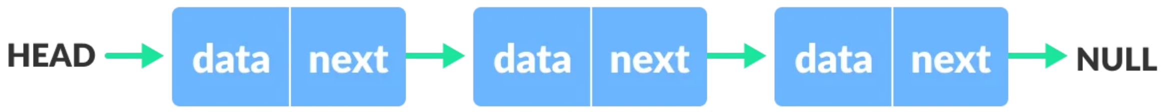
Nama = Helena Jemima Widjaja
NPM = 232310042
Kelas = TI-23-PA1
Matkul = Lab Algoritma Pemrograman & Struktur Data

Rangkuman Linked List

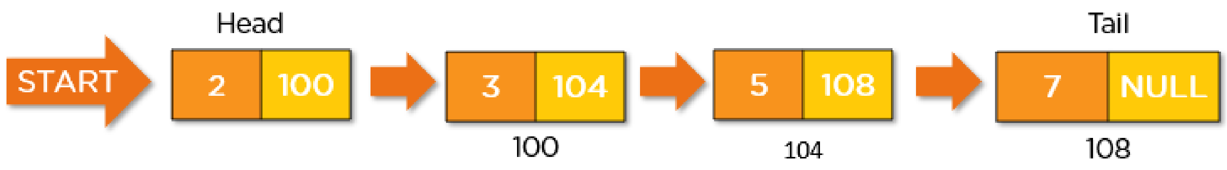
Linked list adalah struktur data linier yang terdiri dari node, dan setiap node berisi data dan pointer ke node berikutnya secara berurutan.



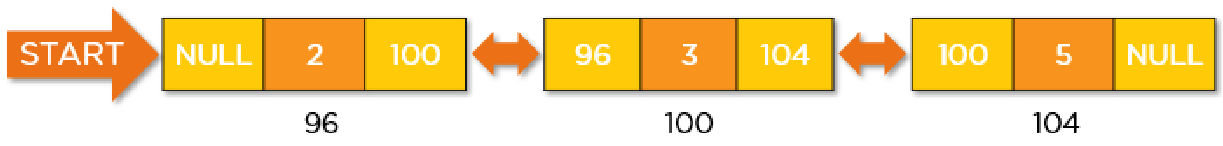
Cara kerja linked list : Elemen pertama dari linked list yang biasa disebut “Head” menyimpan nilai elemen tersebut dan alamat elemen berikutnya, sehingga memiliki pointer ke elemen kedua. Pointer biasanya diberi nama “Next". Elemen kedua juga menyimpan nilai elemen tersebut dan alamat elemen berikutnya. Rantai node ini berlanjut hingga elemen terakhir, di mana ia menyimpan nilai elemen tersebut tetapi penunjuk biasanya menunjuk ke “NULL” yang merupakan cara untuk menunjukkan akhir daftar. Untuk mengakses semua elemen dalam linked list, harus dimulai mulai dari “Head” dan ikuti pointers melalui setiap node hingga mencapai akhir dari list. Proses ini dikenal sebagai traversal.

Jenis - Jenis Linked List :

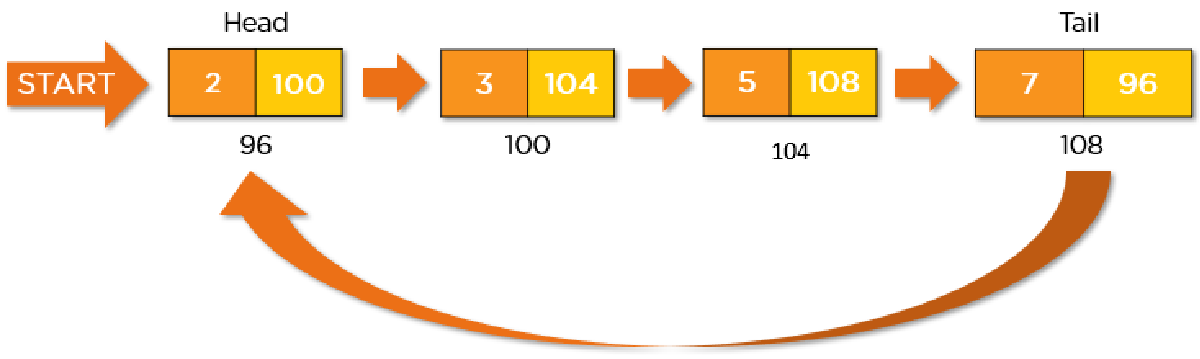
1. Singly Linked List : tipe linked list di mana setiap node berisi data/value dan pointer ke node berikutnya dengan tipe data yang sama. Traversal hanya diperbolehkan dalam satu arah, dari “Head” hingga node terakhir. Diakhiri dengan penunjuk “NULL” di node terakhir.



2. Doubly or Two Way Linked List : tipe linked list di mana setiap node berisi pointer ke node berikutnya dan node sebelumnya secara berurutan. Oleh karena itu, berisi sebuah node memiliki tiga bagian: data/value, pointer ke node berikutnya, dan pointer ke node sebelumnya. Memungkinkan traversal dalam arah maju dan mundur. Berakhir dengan pointer "NULL" di node pertama dan terakhir.

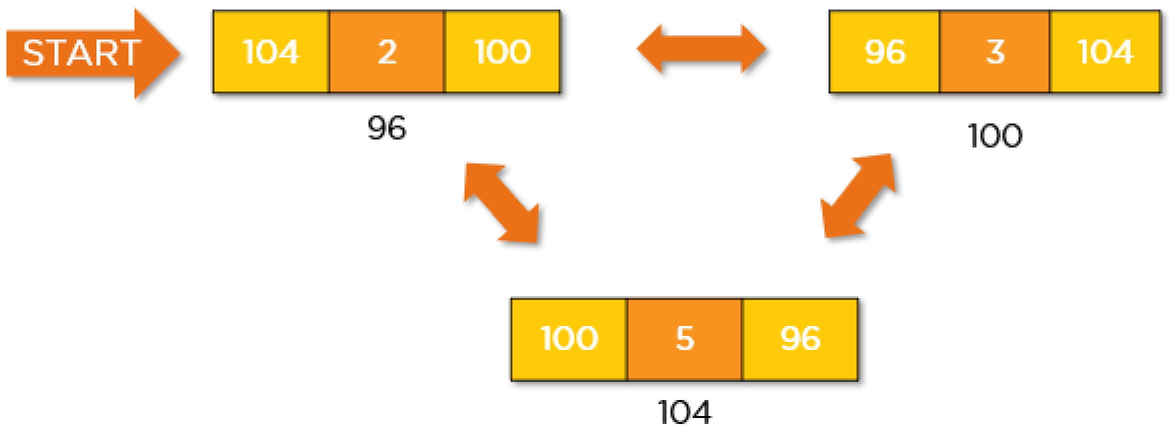


3. Circular Linked List : tipe linked list di mana node terakhir berisi pointer ke node pertama dari list, sehingga menciptakan struktur melingkar. Circular linked list memungkinkan untuk mulai dari node mana pun dan melintasi list ke segala arah, maju dan mundur hingga mencapai node yang sama di mana ia dimulai. Jadi, circular linked list tidak memiliki awal dan akhir yang jelas.



4. Circular Doubly Linked List : tipe linked list yang berisi pointer ke node berikutnya dan node sebelumnya dalam urutan. Perbedaan antara circular doubly linked list dan circular linked list sama dengan perbedaan antara singly linked list dan

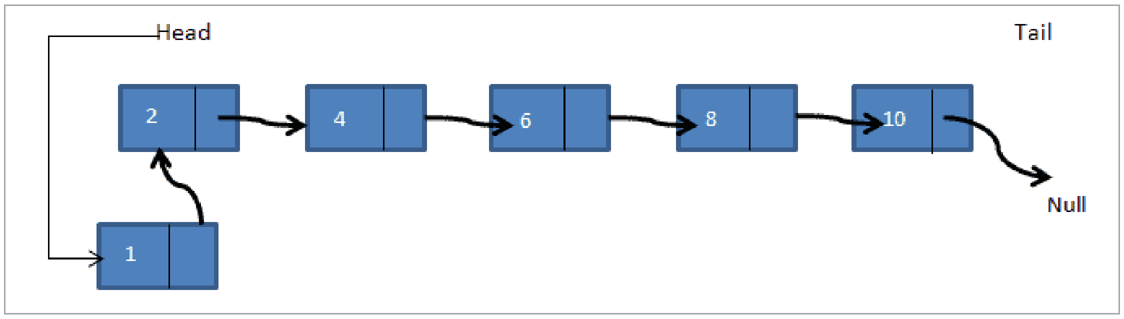
doubly linked list. Circular doubly linked list tidak berisi “NULL” di bidang sebelumnya dari node pertama, sehingga menciptakan struktur melingkar.



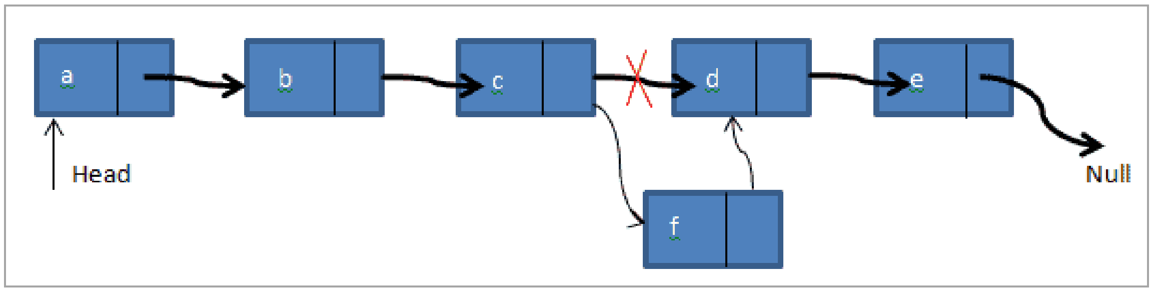
Operasi :

Sama seperti struktur data lainnya, kita juga dapat melakukan berbagai operasi untuk linked list. Namun tidak seperti array, di mana kita dapat mengakses elemen menggunakan subskrip secara langsung meskipun elemen tersebut berada di antara keduanya, kita tidak dapat melakukan akses acak yang sama dengan linked list. Untuk mengakses node mana pun, kita perlu menelusuri linked list dari awal dan baru setelah itu kita dapat mengakses node yang diinginkan.

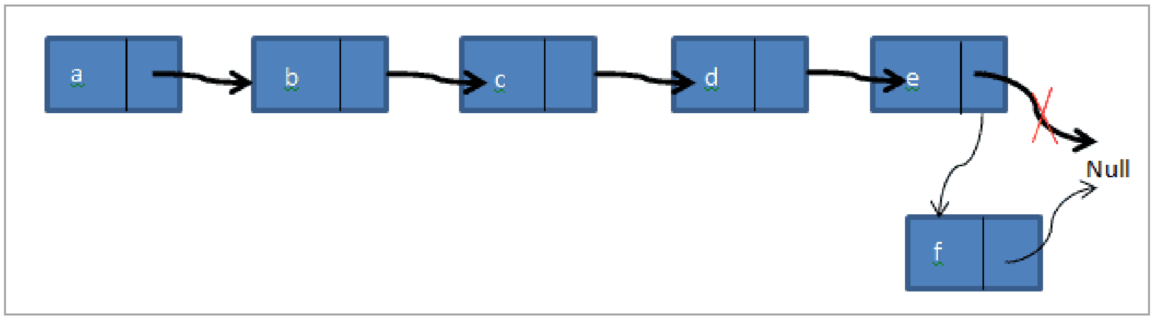
1. Insertion : operasi menambahkan node baru ke linked list yang sudah ada. Setiap kali item data ditambahkan ke linked list/, kita perlu mengubah penunjuk berikutnya dari node sebelumnya dan berikutnya dari item baru yang telah kita sisipkan. Hal kedua yang harus kita pertimbangkan adalah tempat penambahan item data baru. Ada tiga posisi dalam daftar tertaut di mana item data dapat ditambahkan :
    a. At the beginning : jika kita ingin menambahkan node baru sebagai node pertama dari list, maka head yang menunjuk ke node kedua harus diubah menjadi menunjuk ke node baru dan pointer berikutnya dari node baru akan memiliki alamat memori dari node kedua.



b. After the given node : jika kita ingin menambahkan node baru ke list yang sudah ada, kita perlu membuat node baru. Kemudian kita arahkan pointer dari salah satu node yang sudah ada untuk menunjuk ke node baru. Pointer berikutnya dari node baru sekarang menunjuk ke node berikutnya pada nodes list yang diberikan.



c. At the end : jika kita ingin menambahkan node baru di akhir linked list, kita perlu membuat node baru, lalu ubah pointer tail yang menunjuk ke null menjadi menunjuk node baru dan pointer selanjutnya dari node baru tersebut menunjuk ke null.



2. Deletion : menghapus sebuah node dari linked list melibatkan berbagai posisi di mana node tersebut dapat dihapus. Kita dapat menghapus node pertama, node terakhir, atau node acak dari linked list. Setelah penghapusan, kita perlu

menyesuaikan pointer berikutnya dan pointer lainnya dalam linked list dengan tepat.

3. Traversal : mengacu pada proses mengunjungi setiap node dalam linked list untuk melakukan beberapa operasi, mengumpulkan informasi, atau sekadar menampilkan konten. Ini melibatkan perpindahan dari satu node ke node lainnya hingga seluruh linked list telah diproses.
4. Search : melibatkan pencarian nilai atau node tertentu dalam linked list. Untuk mencari nilai apa pun di linked list, kita dapat menelusuri linked list dan membandingkan nilai yang ada di node.
5. Update : melibatkan memodifikasi data yang disimpan dalam sebuah node. Untuk memperbarui nilai node, gunakan operasi search untuk menemukan node yang ingin diperbarui. Setelah node ditemukan, perbarui data yang disimpan di node tersebut.

Aplikasi :

1. Mengimplementasikan Stacks, Queues, Trees, dan Graphs : Dalam stack, elemen ditambahkan dan dihapus dari ujung yang sama (LIFO), sedangkan dalam queues, elemen ditambahkan di salah satu ujung dan dikeluarkan dari ujung lainnya (FIFO). Dalam sebuah tree, setiap node dapat memiliki linked list dari node turunannya. Dalam sebuah graph, linked list dapat digunakan untuk mewakili daftar ketetanggaan, di mana setiap node memiliki linked list dari node - node tetangganya.
2. Mewakili Matriks Rongga: matriks rongga, yang memiliki banyak elemen nol, dapat direpresentasikan menggunakan linked list. Setiap node dalam linked list mewakili elemen bukan nol beserta indeks baris dan kolomnya.
3. Melakukan Operasi Aritmatika pada Bilangan Long Integer: linked list dapat digunakan untuk mengimplementasikan operasi aritmatika pada bilangan long integer, di mana panjang bilangan bulat melebihi kapasitas tipe data standar. Setiap node dalam linked list mewakili satu digit bilangan bulat panjang.

4. Membantu Manajemen Memori: linked list dapat melacak blok memori yang dialokasikan dan bebas, serta mengelola sumber daya memori secara efisien.

Keuntungan dari linked list :

1. Linked list memungkinkan alokasi memori dinamis, memungkinkan list bertambah atau berkurang sesuai kebutuhan dengan mengalokasikan dan membatalkan alokasi memori. Hal ini sangat menguntungkan ketika ukuran struktur data tidak diketahui atau bervariasi.
2. Memasukkan atau menghapus elemen pada linked list dilakukan dengan cara menambah atau menghapus elemen dengan mengatur pointer, tanpa perlu menggeser elemen lainnya.
3. Linked list memungkinkan node memiliki ukuran yang bervariabel, mengakomodasi ukuran data yang berbeda tanpa membuang memori. Hal ini bermanfaat ketika menyimpan elemen dengan panjang yang bervariasi, seperti string.

Kerugian dari linked list :

1. Linked list tidak mendukung akses langsung ke elemen secara random tidak diperbolehkan. Kita hanya dapat mengakses elemen secara berurutan dari bagian atas daftar, mengikuti pointer dari satu node ke node berikutnya. Akses sekuensial ini seringkali kurang efisien dibandingkan akses waktu konstan yang disediakan oleh array. Contohnya, kita tidak dapat mengakses elemen keempat dari linked list secara langsung karena kita tidak tahu di mana letaknya karena elemen tersebut diposisikan secara acak di memori. Kita harus memulai dari awal dan menelusuri daftar hingga kita mencapai elemen keempat, sehingga dapat memakan waktu.
2. Linked list memerlukan lebih banyak memori karena elemen linked list perlu menyimpan 2 hal, nilai elemen itu sendiri dan penunjuk ke elemen berikutnya.

Referensi :

- C++ Tutorial - LINKED LISTS
- Introduction to Linked Lists, Arrays vs Linked Lists, Advantages/Disadvanta...
- https://www.geeksforgeeks.org/advantages-and-disadvantages-of-linked-list/
- https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-linked-list/
- https://www.programiz.com/dsa/linked-list
- https://www.programiz.com/dsa/linked-list-types
- https://www.softwaretestinghelp.com/linked-list/
- https://www.programiz.com/dsa/linked-list-operations
- https://afteracademy.com/blog/types-of-linked-list-and-operation-on-linked-list/

==

A linked list is a linear data structure made up of nodes, and each node contains data and a reference (or pointer) to the next node in the sequence. The pointer points to the next struct or class in its list as it gets added. Unlike an array where we have to statically set the number of elements we need, linked lists allow dynamic memory allocation. Nodes can be allocated and deallocated as needed, which is one of the main advantages of linked lists over fixed-size arrays. (The elements of a linked list are randomly positioned in your memory vs arrays where it stores elements one after the other)

How linked lists work : The first element of a linked list is called “Head” which stores the value of that element and the address of the next element, so it has a pointer to the second element. The pointer usually gets called “Next” to give them a descriptive name as to what they’re pointing to, but they can be anything. The second element also stores the value of that element and the address of the next element. This chain of nodes continues until the last element, where it stores the value of that element but the pointer typically points to null, thereby indicating the end of the list. To access all the elements in a linked list, you start from the Head and follow the pointers through each node until you reach the end of the list. This process is known as traversal.

Types of linked list :

1. Singly Linked List : a linked list where each node contains data/value and a pointer to the next node of the same data type. The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. Traversal is allowed only in one direction, from the head to the last node. Terminates with a null pointer in the last node.
2. Doubly or Two Way Linked List: a linked list where each node contains a pointer to the next as well as the previous node in sequence. Therefore, it contains three parts : data, a pointer to the next node, and a pointer to the previous node. Allows traversal in both forward and backward directions. Terminates with null pointers in the first and last nodes.
3. Circular Linked List: a linked list where the last node contains the pointer to the first node of the list, creating a circular structure. While traversing a circular linked list, one can begin at any node and traverse the list in any direction forward and backward until reaching the same node where it started. Thus, a circular linked list has no distinct beginning or end.

4. Circular Doubly Linked List: a type of linked-list that contains a pointer to the next as well as the previous node in the sequence. The difference between the doubly linked and circular doubly list is the same as that between a singly linked list and a circular linked list. The circular doubly linked list does not contain null in the previous field of the first node, creating a circular structure.

Operations :
Just like the other data structures, we can perform various operations for the linked list as well. But unlike arrays, in which we can access the element using subscript directly even if it is somewhere in between, we cannot do the same random access with a linked list. In order to access any node, we need to traverse the linked list from the start and only then can we access the desired node. Hence accessing the data randomly from the linked list proves to be expensive.

1. Insertions : Insertion operation of linked list adds an item to the linked list. Though it may sound simple, given the structure of the linked list, we know that whenever a data item is added to the linked list, we need to change the next pointers of the previous and next nodes of the new item that we have inserted. The second thing that we have to consider is the place where the new data item is to be added. There are three positions in the linked list where a data item can be added.
    a. At the beginning : if we want to add a new node as the first node of the list, then the head pointing to the second node will now point to the new node and the next pointer of the new node will have a memory address of the second node.
    b. After the given node : a node is given and we want to add a new node after the given node. If we want to add a new node to an already existing list, create a new node. Then we point the next pointer of one of the given nodes to point to the new node. The next pointer of the new node now points to the next node on the given nodes list.

   c. At the end : if we want to add a new node at the end of the linked list, create a new node, then change the tail pointer pointing to null to the new node and the next pointer of the new node is pointed to null.

2. Deletion : deleting a node from a linked list involves various positions from where the node can be deleted. We can delete the first node, last node or a random node from the linked list. After deletion, we need to adjust the next pointer and the other pointers in the linked list appropriately so as to keep the linked list intact.
3. Traversal : refers to the process of visiting each node in the list to perform some operation, gather information, or simply display the contents. It involves moving from one node to another until the entire linked list has been processed.

Applications :

Please change this so it only has 1 short sentence explanations

1. Used to Implement Stacks, Queues, Trees, and Graphs : In a stack, elements are added and removed from the same end (last-in, first-out), while in a queue, elements are added at one end and removed from the other (first-in, first-out). In a tree, each node can have a linked list of its child nodes. In a graph, linked lists can be used to represent adjacency lists, where each vertex has a linked list of its neighboring vertices.
2. Representing Sparse Matrices: Sparse matrices, which have a large number of zero elements, can be efficiently represented using linked lists. Each node in the linked list represents a non-zero element along with its row and column indices.
3. Performing Arithmetic Operations on Long Integers: Linked lists are used to implement arithmetic operations on long integers, where the length of the integer exceeds the capacity of standard data types. Each node in the linked list represents a digit of the long integer.
4. Memory Management: Linked lists are valuable in memory management scenarios, keeping track of allocated and free memory blocks, and efficiently managing memory resources.

Advantages of linked list :

1. Linked lists allow dynamic allocation of memory, enabling the list to grow or shrink as needed by allocating and deallocating memory. This is particularly advantageous when the size of the data structure is unknown or varies.
2. Inserting or deleting elements in a linked list is done by adding or removing elements by adjusting the pointers, without the need to shift other elements.
3. Linked lists allow nodes to have variable sizes, accommodating different data sizes without wasting memory. This is particularly beneficial when storing elements of varying lengths, such as strings.

Disadvantages of linked lists :

1. Linked lists don't support direct access to arbitrary elements by index is not allowed. You can only access elements sequentially from the head of the list, following the pointers from one node to the next. This sequential access is often less efficient than the constant-time access provided by arrays. For example, you can't access the fourth element of a linked list directly because you don't know where it is due to it being positioned randomly in your memory. You need to start from the head and traverse the list until you reach the fourth element, making it time consuming.
2. Linked lists need more memory because the elements of linked list needs to store 2 things, the value of the element itself and a pointer to the next element

A node of a linked list consists of two parts : a value and a pointer to the next element, this means you can't represent an element of a linked list with a built in data type, but you need to create a user defined data type in order to represent an element of a linked list.

1. You need to use class or struct in order to represent one element of a linked list. You can name it to be “Node” or “Element”. Inside the class or struct, you put two parts of the node; the value of the node and a pointer to the next node.

2. In the main part of your code, create a pointer to the next element, usually called "Head", and create more depending on how many elements you want by changing the name to "two, three" and so on.

Linked lists can be used to improve the performance of algorithms that need to frequently insert or delete items from large collections of data.
Implementing algorithms such as the LRU cache, which uses a linked list to keep track of the most recently used items in a cache.