PENJELASAN MATERI-4

Double Linked List

Dosen pengampu:
Randi Proska Sandra, S.Pd, M.Sc.

Disusun oleh:
Muhammad Ghazian Tsaqif Zhafiri Andoz (23343057)

NO	Baris Program	Petikan Source Code	Penjelasan

1	Percobaan 1
	4-8
		`struct Node {`
		`  int data;`
		`  struct Node* next;  //`
		`Pointer ke node selanjutnya`
		`  struct Node* prev;  //`
		`Pointer ke node sebelumnya`
		`};`
			struct Node digunakan untuk merepresentasikan sebuah simpul (node) dalam suatu linked list. Setiap simpul memiliki dua bagian utama: **data**, yang menyimpan nilai data, dan dua pointer: **next**, yang menunjukkan ke simpul berikutnya, dan **prev**, yang menunjukkan ke simpul sebelumnya.

2	Percobaan 1
	11
		`struct Node* new_node =`
		`(struct`
		`Node*)malloc(sizeof(struct`
		`Node));`
			malloc(sizeof(struct Node)) mengalokasikan memori untuk satu node baru, dan **(struct Node*)** mengonversi alamat memori yang dialokasikan menjadi tipe pointer ke struct Node.

3	Percobaan 1
	13-15
		`new_node->data = new_data;`
		`  new_node->next =`
		`(*head_ref);`
		`  new_node->prev = NULL;`
			Baris-baris tersebut menginisialisasi node baru dengan data baru, menetapkan node baru ke depan linked list, dan menetapkan node baru tanpa node sebelumnya (**NULL**).

4	Percobaan 1
	17
		`  if ((*head_ref) != NULL)`
		`  (*head_ref)->prev =`
		`new_node;`
		
		`  (*head_ref) = new_node;`
			Jika **head_ref** tidak kosong, atur **prev** dari **head_ref** ke **new_node**. Kemudian, perbarui **head_ref** menjadi **new_node**.

5	Percobaan 1
	38-45
		`int main() {`
		`  struct Node* head = NULL;`
		`  push(&head, 1);`
		`  push(&head, 2);`
		`  push(&head, 3);`
		`  printList(head);`
		`  return 0;`
		`}`
			Sebuah pointer **head** diinisialisasi dengan **NULL**. Tiga elemen (**1, 2, 3**) dimasukkan menggunakan fungsi **push**. Fungsi **printList** dipanggil untuk mencetak isi linked list ke depan dan belakang. Program mengembalikan nilai 0 untuk mengakhiri.

6	Percobaan 2 10	`void push(struct Node**` `head_ref, int new_data)`	Fungsi **push** mengambil alamat dari (**head_ref**) dan data baru (**new_data**) sebagai argumen. Ini menambahkan sebuah node baru dengan data **new_data** di awal linked list, memindahkan head linked list untuk menunjuk ke node baru, dan memperbarui

			pointer **prev** dari node sebelumnya jika ada.

7	Percobaan 2
	22-26
		`if ((*head_ref) != NULL)`
		`(*head_ref)->prev = new_node;`
		`(*head_ref) = new_node;`
			Memeriksa apakah (***head_ref**) tidak **NULL**. Jika tidak **NULL**, itu berarti head tidak kosong, dan kemudian **prev** dari **head** sekarang ditetapkan sebagai node baru (**new_node**). Kemudian, head diperbarui untuk mengarah ke node baru tersebut.

8	Percobaan 2 37	`struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));`	Fungsi malloc mengalokasikan ruang memori sebesar **sizeof(struct Node)** untuk node baru. Kemudian, pointer **new_node** mengarah ke lokasi memori yang baru dialokasikan tersebut, dengan tipe data yang sesuai yaitu **struct Node***.

9	Percobaan 2
	40-49
		`new_node->data = new_data;`
		`new_node->next = prev_node->next;`
		`prev_node->next = new_node;`
		`new_node->prev = prev_node;`
			1. Set nilai data pada node baru.
			2. Sambungkan node baru ke node setelah **prev_node**.
			3. Sambungkan **prev_node** ke node baru.
			4. Atur node sebelumnya dari node baru.

10	Percobaan 2
	58-63
		`printf("[ Traversal ke arah depan ]\n~> ");`
		`while (node != NULL) {`
		`printf(" %d ", node->data);`
		`last = node;`
		`node = node->next;`
		`}`
			Mencetak traversal ke arah depan dengan menampilkan data dari setiap node. Loop akan terus berjalan selama node tidak **NULL**. Setiap **iterasi**, program mencetak data dari node saat ini (**node->data**), kemudian memindahkan node ke node berikutnya dalam linked list (**node = node->next**).

11	Percobaan 3 27	`void append(struct Node** head_ref, int new_data) {`	1. Alokasi memori untuk node baru. 2. Mengatur data node baru dengan data yang diberikan. 3. Mengatur pointer next node baru sebagai **NULL**, karena node baru akan menjadi node terakhir. 4. Jika linked list kosong, maka node baru menjadi head.

			5. Jika tidak kosong, **iterasikan** sampai akhir linked list. 6. Mengatur pointer next dari node terakhir untuk menunjuk ke node baru. 7. Mengatur pointer prev dari node baru untuk menunjuk ke node terakhir.
12	Percobaan 3 29-30	`struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));` `struct Node* last = *head_ref;`	Membuat pointer baru ke Node dengan alokasi memori menggunakan **malloc()**. Pointer ini menunjuk ke node terakhir dalam linked list yang diakses melalui **head_ref**, membantu menambahkan node baru ke akhir.

13	Percobaan 3
	4-8
		`struct Node {`
		`int data;`
		`struct Node* next;`
		`struct Node* prev;`
		`};`
			Menyimpan data bertipe integer dan memiliki dua pointer: Memungkinkan navigasi maju (**next**) dan mundur (**prev**).

14	Percobaan 3 25	`(*head_ref) = new_node;`	Mengubah alamat memori yang ditunjuk oleh **head_ref** sehingga menunjuk ke node baru yang telah dibuat, sehingga membuat node baru tersebut menjadi head dari linked list.

15	Percobaan 3
	73-78
		`append(&head, 11);`
		`push(&head, 22);`
		`push(&head, 14);`
		`append(&head, 98);`
		
		`printList(head);`
			• **append(&head, 11)** : Menambahkan node baru dengan data 11 di akhir linked list.
			• **push(&head, 22)** : Menambahkan node baru dengan data 22 di awal linked list.
			• **push(&head, 14)** : Menambahkan node baru dengan data 14 di awal linked list.
			• **append(&head, 98)** : Menambahkan node baru dengan data 98 di akhir linked list.
			• **printList(head)** : Mencetak isi linked list dari awal ke akhir dan dari akhir ke awal.

16	Percobaan 4
	12-14
		`new_node->data = new_data;`
		`    new_node->next = (*head_ref);`
		`    new_node->prev = NULL;`
			• Mengatur nilai data dari node baru (**new_node->data**) menjadi nilai **new_data** yang diberikan.
			• Menetapkan next dari node baru (**new_node->next**) ke alamat node yang saat ini berada di depan daftar head (***head_ref**).
			• Menetapkan prev dari node baru (**new_node->prev**) sebagai **NULL**, karena node baru akan menjadi node pertama.

17	Percobaan 4 51	`void printList(struct Node* node)`	Mengambil pointer ke node pertama dalam daftar dan mencetak nilai dari setiap node secara berurutan, terlebih dahulu dari depan ke belakang, dan kemudian dari belakang ke depan.

18	Percobaan 4
	15-17
		`if ((*head_ref) != NULL)`
		`        (*head_ref)->prev = new_node;`
		`    (*head_ref) = new_node;`
			Memperbarui pointer **prev** dari node yang sebelumnya menjadi head dengan menunjukkannya ke node baru (**new_node**). Kemudian, mengubah pointer head untuk menunjuk ke node baru (**new_node**), sehingga node baru tersebut menjadi head yang baru.

19	Percobaan 4 20	`void insertBefore(struct Node** head_ref, struct Node* next_node, int new_data)`	Parameter **head_ref** adalah referensi ke kepala linked list, **next_node** adalah node yang akan menjadi node setelah node baru, dan **new_data** adalah data yang akan dimasukkan ke dalam node baru.
20	Percobaan 4 72	`insertBefore(&head, head->next, 8);`	Menyisipkan node baru dengan data **8** sebelum node yang berada setelah head.