Nomor Program	Baris Program	Petikan Source Code	Penjelasan
1	1-2	`#include <stdio.h>` `#include <stdlib.h>`	stdio.h diperlukan untuk fungsi input-output standar seperti printf() dan scanf(), sedangkan stdlib.h menyediakan fungsi-fungsi umum seperti alokasi memori dinamis (malloc(), calloc(), realloc()) dan konversi string ke angka (atoi(), atof()).

1	5-10	`struct Node`
		`{`
		`int data; struct Node *next; struct Node *prev;`
		`};`
			Deklarasi struktur baru dengan nama node (simpul). Next dan prev adalah variable pointer yang akan digunakan untuk mengarahkan ke simpul sebelum atau setelah sebuah simpul baru dibuat

1	10-25	`void push(struct Node** head_ref, int new_data)`
		`{`
		`struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));`
		
		`new_node->data = new_data;`
		`new_node->next = (*head_ref);`
		`new_node->prev = NULL;`
		
		`if ((*head_ref) != NULL)`
		`(*head_ref)->prev = new_node;`
		
		`(*head_ref) = new_node;`
		`}`
			Fungsi ini menggunakan `malloc` untuk mengalokasikan memori untuk node baru dengan ukuran yang sesuai dengan struktur 'Node'. Pointer `next` dari node baru diatur untuk menunjuk ke head saat ini dari linked list (`*head_ref`), sehingga node baru ditempatkan di depan linked list. Pointer `prev` dari node baru diatur ke NULL karena node baru akan menjadi node pertama dalam linked list. Terakhir, pointer head dari linked list (`*head_ref`) diubah untuk menunjuk ke node baru, menjadikannya sebagai head baru dari linked list.

1	26-40	`void printList(struct Node* node)`
		`{`
		`struct Node* last;`
		`printf("\nTraversal in forward direction \n");`
		`while (node != NULL) {`
		`printf(" %d ", node->data);`
		`last = node;`
		`node = node->next;`
		`}`
		`printf("\nTraversal in reverse direction \n");`
		`while (last != NULL) {`
		`printf(" %d ", last->data);`
		`last = last->prev;`
		`}`
		`}`
			Fungsi `printList` mencetak elemen-elemen dari double linked list dalam urutan maju dan mundur. Pertama, mencetak "Traversal in forward direction" dengan loop `while`, mencetak nilai `data` dari setiap node saat bergerak dari head ke tail menggunakan pointer `next`, dan menyimpan node terakhir dalam `last`. Setelah mencapai akhir linked list, mencetak "Traversal in reverse direction" dengan loop `while` kedua, mencetak nilai `data` dari setiap node saat bergerak mundur dari tail ke head menggunakan pointer `prev`, dimulai dari node terakhir yang disimpan dalam `last`. Ini memberikan visualisasi urutan elemen dalam linked list dari awal ke akhir dan sebaliknya.

1	41-52	`int main()`
		`{`
		
		`struct Node* head = NULL;`
		` push(&head, 6);`
		` push(&head, 5);`
		` push(&head, 2);`
		` printf("Created DLL is: ");`
		` printList(head);`
		` getchar();`
		
		` return 0;`
		`}`
			Fungsi `main` membuat linked list awalnya kosong. Tiga node baru ditambahkan dengan nilai 6, 5, dan 2 menggunakan fungsi `push`, menempatkannya di depan linked list sehingga urutannya menjadi [2, 5, 6]. Setelah penambahan node, pesan "Created DLL is:" dicetak, diikuti dengan pemanggilan `printList` untuk mencetak elemen-elemen linked list dalam dua arah: maju (dari head ke tail) dan mundur (dari tail ke head). Program menunggu input pengguna sebelum berakhir menggunakan `getchar()`, dan mengembalikan nilai 0 untuk menunjukkan keberhasilan program.

2	4-9	`struct Node`
		`{`
		`int data;`
		`struct Node *next;`
		`struct Node *prev;`
			Baris program ini mendefinisikan struktur `Node` untuk linked list ganda. Struktur ini terdiri dari integer `data`, dan dua pointer: `next` yang menunjuk ke node berikutnya, dan `prev` yang menunjuk ke node sebelumnya. Dengan definisi ini, linked list ganda memungkinkan traversal dua arah melalui linked list.

2	10-28	`void push(Node** head_ref, int new_data)`
		`{`
		`Node* new_node = new Node();`
		`new_node->data = new_data;`
		`new_node->next = (*head_ref);`
		`new_node->prev = NULL;`
		
		`if ((*head_ref) != NULL)`
		`(*head_ref)->prev = new_node;`
		
		`(*head_ref) = new_node;`
		`}`
			Program ini adalah implementasi dari fungsi `push` untuk menambahkan node baru di depan linked list. Fungsi ini menggunakan `new` untuk mengalokasikan memori untuk node baru, memasukkan data baru, dan mengatur pointer `next` untuk menunjuk ke head saat ini. Jika head tidak NULL, pointer `prev` dari head diubah untuk menunjuk ke node baru. Akhirnya, head dari linked list diperbarui untuk menunjuk ke node baru, menjadikannya head baru. Dengan cara ini, fungsi `push` mempertahankan konsistensi linked list ganda dengan memasukkan node baru di depannya.

2	29-49	`void insertAfter(struct Node* prev_node, int new_data)`
		`{`
		`if (prev_node == NULL) {`
		`printf("the given previous node cannot be NULL");`
		`return;`
		`}`
		`struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));`
		
		`new_node->data = new_data;`
			Program ini mendefinisikan fungsi `insertAfter` untuk menyisipkan node baru setelah node tertentu dalam linked list. Fungsi ini memeriksa apakah `prev_node` NULL. Jika iya, menampilkan pesan kesalahan karena tidak mungkin menyisipkan node setelah node NULL. Jika tidak, fungsi menggunakan `malloc` untuk mengalokasikan memori untuk node baru, mengisi data baru, dan mengatur pointer `next` untuk menunjuk ke node berikutnya dari `prev_node`. Pointer `next` dari `prev_node` diubah untuk menunjuk ke node

		`new_node->next = prev_node->next;`
		`prev_node->next = new_node;`
		`new_node->prev = prev_node;`
		
		`if (new_node->next != NULL)`
		`new_node->next->prev = new_node;`
		`}`
			baru. Node baru dihubungkan ke `prev_node` dengan mengatur pointer `prev`-nya, dan jika node baru memiliki node berikutnya, pointer `prev` dari node berikutnya juga diubah untuk menunjuk ke node baru. Dengan cara ini, fungsi `insertAfter` memasukkan node baru ke dalam linked list setelah node tertentu dan memperbarui pointer yang relevan untuk memastikan integritas linked list ganda.

2	50-64	`void printList(struct Node* node)`
		`{`
		`struct Node* last;`
		`printf("\nTraversal in forward direction \n");`
		`while (node != NULL) {`
		`printf(" %d ", node->data);`
		`last = node;`
		`node = node->next;`
		`}`
		`printf("\nTraversal in reverse direction \n");`
		`while (last != NULL) {`
		`printf(" %d ", last->data);`
		`last = last->prev;`
		`}`
		`}`
			Fungsi `printList` mencetak elemen-elemen linked list dalam urutan maju dan mundur. Pertama, mencetak elemen-elemen dalam urutan maju dengan loop `while`, menyimpan node terakhir dalam `last`. Setelah mencapai akhir linked list, mencetak elemen-elemen dalam urutan mundur dengan loop `while` kedua menggunakan pointer `prev`. Ini memberikan visualisasi kedua arah dari linked list dalam satu fungsi.

2	65-77	`int main()`
		`{`
		`/* Start with the empty list */`
		`struct Node* head = NULL;`
		`push(&head, 6);`
		`push(&head, 5);`
		`push(&head, 2);`
		`insertAfter(head->next, 5);`
		`printf("Created DLL is: ");`
		`printList(head);`
		`getchar();`
		`return 0;`
		`}`
			Fungsi `main` menginisialisasi linked list kosong, menambahkan tiga node baru dengan nilai 6, 5, dan 2 menggunakan `push`, dan menyisipkan node baru dengan nilai 5 setelah node kedua menggunakan `insertAfter`. Hasilnya dicetak dengan `printList` untuk visualisasi urutan maju dan mundur. Program menunggu input pengguna sebelum berakhir, mengembalikan nilai 0 untuk menunjukkan keberhasilan.

3	5-9	`struct Node {`
		`int data;`
		`struct Node *next;`
		`struct Node *prev;`
		`};`
			Potongan program ini mendefinisikan struktur `Node` untuk linked list ganda. Struktur ini memiliki tiga anggota: `data` untuk menyimpan nilai, dan dua pointer `next` dan `prev` untuk menunjuk ke node berikutnya dan sebelumnya dalam linked list. Dengan definisi ini, linked list ganda memungkinkan traversal dua arah.

3	11-24	`void push(struct Node **head_ref, int new_data) {`	Fungsi `push` menambahkan node baru di depan linked list. Pertama, memori dialokasikan untuk node baru, lalu data

		struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
		new_node->data = new_data;
		
		new_node->next = *head_ref;
		new_node->prev = NULL;
		if (*head_ref != NULL)
		(*head_ref)->prev = new_node;
		*head_ref = new_node;
		}
			dimasukkan. Node baru diatur dengan pointer `next` menunjuk ke head saat ini, dan `prev` ke NULL. Jika head tidak NULL, pointer `prev` dari head diubah untuk menunjuk ke node baru. Akhirnya, head dari linked list diperbarui untuk menunjuk ke node baru, menjadikannya head baru.

3	26-47	void append(struct Node **head_ref, int new_data) {
		
		struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
		struct Node *last = *head_ref;
		new_node->data = new_data;
		new_node->next = NULL;
		
		if (*head_ref == NULL) {
		new_node->prev = NULL;
		*head_ref = new_node;
		return;
		}
		while (last->next != NULL)
		last = last->next;
		last->next = new_node;
		new_node->prev = last;
		}
			Fungsi `append` menambahkan node baru di akhir linked list. Pertama, memori dialokasikan untuk node baru dan data dimasukkan. Node baru diatur dengan pointer `next` ke NULL. Jika linked list kosong, node baru dijadikan sebagai head. Jika tidak, fungsi mencari node terakhir dalam linked list dan mengaitkan node baru sebagai node berikutnya, serta memperbarui pointer `prev` dari node baru untuk menunjuk ke node terakhir sebelumnya.

3	49-62	void printList(struct Node *node) {
		struct Node *last = NULL;
		printf("\nTraversal in forward direction \n");
		while (node != NULL) {
		printf(" %d ", node->data);
		last = node;
		node = node->next;
		}
		printf("\nTraversal in reverse direction \n");
		while (last != NULL) {
		printf(" %d ", last->data);
		last = last->prev;
		}
		}
			Fungsi `printList` mencetak elemen-elemen linked list dalam urutan maju dan mundur. Pertama, mencetak elemen dalam urutan maju dengan loop `while`, menyimpan node terakhir dalam variabel `last`. Setelah mencapai akhir linked list, mencetak elemen dalam urutan mundur dengan loop `while` kedua menggunakan pointer `prev`. Ini memberikan visualisasi kedua arah dari linked list dalam satu fungsi.

3	64-79	int main() {
		struct Node *head = NULL;
		append(&head, 6);
		push(&head, 7);
		push(&head, 1);
		append(&head, 4);
		printf("Created DLL is: ");
		printList(head);
		getchar();
		return 0;
			Fungsi `main` ini menginisialisasi linked list kosong dan kemudian menambahkan beberapa node ke dalamnya. Node-node tersebut dimasukkan di awal dan di akhir linked list menggunakan fungsi `push` dan `append`. Setelah node-node dimasukkan, fungsi `printList` dipanggil untuk mencetak linked list dalam urutan maju dan mundur. Akhirnya,

		} 	program menunggu input pengguna sebelum berakhir.

4	4-9	struct Node
		{
		int data;
		struct Node *next;
		struct Node *prev;
		};
			Mendefinisikan struktur `Node` untuk linked list ganda. Setiap node memiliki tiga anggota: `data` untuk menyimpan nilai, `next` untuk menunjuk ke node berikutnya, dan `prev` untuk menunjuk ke node sebelumnya. Dengan definisi ini, linked list ganda memungkinkan traversal dua arah.

4	11-29	void push(struct Node** head_ref, int new_data)
		{
		
		struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
		
		new_node->data = new_data;
		new_node->next = (*head_ref);
		new_node->prev = NULL;
		(*head_ref)->prev = new_node;
		(*head_ref) = new_node;
		}
			`push` menambahkan node baru di depan linked list. Pertama, alokasi memori untuk node baru. Data baru dimasukkan ke dalamnya. Pointer `next` dari node baru menunjuk ke head saat ini, sementara `prev` diatur NULL karena node baru akan menjadi head. Jika head tidak NULL, pointer `prev` dari head diubah. Head dari linked list diperbarui untuk menunjuk ke node baru. Ini memastikan node baru dimasukkan di depan linked list.

4	30-50	void insertAfter(struct Node* prev_node, int new_data)
		{
		
		if (prev_node == NULL) {
		printf("the given previous node cannot be NULL");
		return;
		}
		struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
		
		new_node->data = new_data;
		new_node->next = prev_node->next;
		prev_node->next = new_node;
		new_node->prev = prev_node;
		if (new_node->next != NULL)
		new_node->next->prev = new_node;
		}
			`insertAfter` menyisipkan node baru setelah node tertentu dalam linked list. Pertama, fungsi memeriksa apakah node sebelumnya yang diberikan tidak NULL. Jika ya, fungsi mengalokasikan memori untuk node baru dan mengatur data baru. Node baru dihubungkan ke node sebelumnya dan ke node berikutnya. Fungsi juga memperbarui pointer `prev` dari node berikutnya jika ada.

4	51-65	void printList(struct Node* node)
		{
		struct Node* last;
		printf("\nTraversal in forward direction \n");
		while (node != NULL) {
		printf(" %d ", node->data);
		last = node;
		node = node->next;
		}
		printf("\nTraversal in reverse direction \n");
			`printList` mencetak elemen-elemen linked list dalam urutan maju dan mundur. Pertama, elemen-elemen dicetak dalam urutan maju menggunakan loop `while`, sambil menyimpan node terakhir dalam variabel `last`. Setelah mencapai akhir linked list, elemen-elemen dicetak dalam urutan mundur dengan loop `while` kedua menggunakan pointer `prev`. Ini memberikan visualisasi kedua arah dari linked list dalam satu fungsi.

		while (last != NULL) {
		printf(" %d ", last->data);
		last = last->prev;
		}
		}	

4	66-78	int main()
		{
		/* Start with the empty list */
		struct Node* head = NULL;
		push(&head, 6);
		push(&head, 5);
		push(&head, 2);
		insertAfter(head->next, 5);
		printf("Created DLL is: ");
		printList(head);
		getchar();
		return 0;
		}
			`main` membuat linked list kosong. Tiga node baru ditambahkan di depan dengan nilai 6, 5, dan 2. Node baru dengan nilai 5 disisipkan setelah node kedua. Hasil linked list dicetak, lalu program menunggu input sebelum berakhir.