Tugas Jobsheet 4 Doubly Linked List

Nama : Muhamad Akbar Fauzan
NIM : 23343075

Nomor Program	Baris Program	Petikan Souce Code	Penjelasan

1	5-10	struct Node
		{
		int data;
		struct Node *next; // Pointer to next node
		struct Node *prev; // Pointer to previous node
		};
			Baris program ini merupakan fondasi dari representasi data dalam Doubly Linked List. Dengan menggunakan struktur Node yang didefinisikan dengan pointer next dan prev, setiap simpul dalam Doubly Linked List dapat saling terhubung baik ke simpul berikutnya maupun sebelumnya, memungkinkan navigasi maju dan mundur dalam struktur data tersebut.

	12-26	void push(struct Node** head_ref, int new_data)
		{
		/* 1. allocate node */
		struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
		/* 2. put in the data */
		new_node->data = new_data;
		/* 3. Make next of new node as head and previous as NULL */
		new_node->next = (*head_ref);
		new_node->prev = NULL;
		/* 4. change prev of head node to new node */
		if ((*head_ref) != NULL)
		(*head_ref)->prev = new_node;
		/* 5. move the head to point to the new node */
		(*head_ref) = new_node;
		}
			Fungsi push ini digunakan untuk menambahkan node baru ke awal daftar tertentu. Node baru disimpan dalam variabel new_node dan diinisialisasi dengan data dan referensi ke node selanjutnya. Jika daftar tertentu kosong, new_node menjadi head dari daftar tertentu. Jika daftar tertentu sudah memiliki node, new_node menjadi anak pertama dan node yang sebelumnya menjadi anak pertama menjadi anak kedua dari new_node. Selain itu, new_node juga disimpan dalam variabel head_ref untuk mengganti head dari daftar tertentu.

	28-36	void printList(struct Node* node)
		{
		struct Node* last;
		printf("\nTraversal in forward direction \n");
		while (node != NULL) {
		printf(" %d ", node->data);
		last = node;
			Fungsi printList digunakan untuk mencetak data node dari suatu linked list, dimana linked list tersebut memiliki dua arah navigasi, yaitu prev (sebelumnya) dan next (selanjutnya).
			
			program akan melakukan traversal pada linked list dengan arah normal (dari head ke tail), yaitu dengan cara

		node = node->next;
		}
		
			mengambil data node selanjutnya dari node yang sedang ditunjuk oleh pointer node, sampai pointer node menunjuk ke node NULL (akhir dari linked list). Setiap kali melakukan traversal, data node yang sedang ditunjuk oleh pointer node akan dicetak. Setelah traversal dengan arah normal selesai, pointer last akan diset untuk menunjuk ke node terakhir yang telah ditemui.

	38-43	printf("\nTraversal in reverse direction \n");
		while (last != NULL) {
		printf(" %d ", last->data);
		last = last->prev;
		}
		}
			Program akan melakukan traversal pada linked list dengan arah terbalik (dari tail ke head), yaitu dengan cara mengambil data node sebelumnya dari node yang sedang ditunjuk oleh pointer last, sampai pointer last menunjuk ke node NULL (awal dari linked list). Setiap kali melakukan traversal, data node yang sedang ditunjuk oleh pointer last akan dicetak.

	45-56	int main()
		{
		/* Start with the empty list */
		struct Node* head = NULL;
		push(&head, 6);
		push(&head, 5);
		push(&head, 2);
		printf("Created DLL is: ");
		printList(head);
		getchar();
		return 0;
		}
			Fungsi push digunakan untuk memasukkan sebuah node baru ke dalam linked list. Dalam hal ini, fungsi ini dipanggil tigakali dengan nilai yang dimasukkan adalah 2, 5, dan 6. Nilai tersebut akan dimasukkan sebagai data dari node baru, sedangkan pointer ke node sebelumnya akan disimpan pada atribut prev dan pointer ke node selanjutnya akan disimpan pada atribut next.
			Fungsi printList digunakan untuk mencetak isi dari linked list. Fungsi ini akan memulai dari node awal (head) dan mencetak data dari node tersebut. Setelah itu, fungsi akan melanjutkan ke node selanjutnya dengan menggunakan atribut next hingga selesai mencetak seluruh isi linked list.
			Selain itu, program ini juga menggunakan perintah getchar() sebelum return 0, yang digunakan untuk menunda tampilan hasil output hingga tombol enter ditekan.

2	5-10	struct Node	Bagian program ini adalah struktur

		{
		int data;
		struct Node *next; // Pointer to next node
		struct Node *prev; // Pointer to previous node
		};
			dari node untuk representasi sebuah doubly linked list (DLL) di C. Setiap node pada DLL memiliki tiga bagian: data, pointer ke node berikutnya (next), dan pointer ke node sebelumnya (prev). Hal ini memungkinkan traversal dari DLL dalam arah depan dan belakang.

	11-29	void push(Node** head_ref, int new_data)
		{
		/* 1. allocate node */
		Node* new_node = new Node();
		
		/* 2. put in the data */
		new_node->data = new_data;
		
		/* 3. Make next of new node as head and previous as NULL */
		new_node->next = (*head_ref);
		new_node->prev = NULL;
		
		/* 4. change prev of head node to new node */
		if ((*head_ref) != NULL)
		(*head_ref)->prev = new_node;
		
		/* 5. move the head to point to the new node */
		(*head_ref) = new_node;
		}
			Bagian program ini adalah implementasi dari fungsi push yang digunakan untuk menambahkan elemen baru ke depan (atau awal) dari sebuah linked list dengan representasi ganda (doubly linked list). Pertama, sebuah node baru dialokasikan menggunakan operator new dan diisi dengan data yang diberikan. Kemudian, next dari node baru diarahkan ke node yang saat ini menjadi kepala (head) dari linked list, sementara prev-nya diatur menjadi NULL karena node baru akan menjadi head sehingga tidak memiliki node sebelumnya. Jika linked list tidak kosong, maka prev dari node yang sebelumnya adalah head akan diubah untuk menunjuk ke node baru. Akhirnya, head dari linked list diperbarui untuk menunjuk ke node baru yang baru saja ditambahkan. Dengan demikian, prosedur push ini memungkinkan penambahan elemen baru ke depan dari linked list dengan langkah-langkah yang jelas dan terstruktur.

	30-50	void insertAfter(struct Node* prev_node, int new_data)
		{
		/*1. check if the given prev_node is NULL */
		if (prev_node == NULL) {
		printf("the given previous node cannot be NULL");
		return;
		}
		/* 2. allocate new node */
		struct Node* new_node = (struct Node*)malloc(sizeof(struct
			Fungsi `insertAfter` dalam program ini bertujuan untuk menyisipkan node baru setelah node tertentu dalam linked list. Pertama, program memeriksa apakah node sebelumnya yang diberikan (`prev_node`) tidak kosong. Jika kosong, artinya tidak dapat melanjutkan proses, sehingga program mencetak pesan kesalahan dan berhenti. Selanjutnya, program membuat node baru dengan alokasi memori dinamis dan mengisi data yang diberikan. Node baru ini akan

		Node));
		/* 3. put in the data */
		new_node->data = new_data;
		/* 4. Make next of new node as next of prev_node */
		new_node->next = prev_node->next;
		/* 5. Make the next of prev_node as new_node */
		prev_node->next = new_node;
		/* 6. Make prev_node as previous of new_node */
		new_node->prev = prev_node;
		/* 7. Change previous of new_node's next node */
		if (new_node->next != NULL)
		new_node->next->prev = new_node;
		}
			ditempatkan setelah `prev_node`. Pointer `next` dari node baru diarahkan ke node yang sebelumnya merupakan `next` dari `prev_node`, sehingga node baru tersisip di antara keduanya. Pointer `next` dari `prev_node` diperbarui untuk menunjuk ke node baru. Selain itu, pointer `prev` dari node baru diarahkan ke `prev_node`, memastikan konsistensi struktur linked list. Terakhir, jika node baru tidak berada di ujung linked list, pointer `prev` dari node setelah node baru juga diubah untuk menunjuk kembali ke node baru. Dengan langkah-langkah ini, fungsi ini memungkinkan penambahan node baru dengan mudah dan efisien setelah node tertentu dalam linked list.

	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");
		while (last != NULL) {
		printf(" %d ", last->data);
		last = last->prev;
		}
		}
			Bagian program ini mendefinisikan fungsi `printList` yang bertujuan untuk mencetak isi linked list baik secara maju (forward) maupun mundur (reverse). Pertama-tama, program menyiapkan sebuah pointer `last` untuk menyimpan alamat terakhir dari linked list. Selanjutnya, program memulai iterasi maju (forward traversal) dari awal linked list. Selama iterasi maju, setiap data dari node dicetak, dan alamat node tersebut disimpan dalam `last`. Setelah mencapai akhir linked list (node terakhir adalah NULL), program beralih ke iterasi mundur (reverse traversal) dengan menggunakan pointer `last` yang menyimpan alamat terakhir node. Selama iterasi mundur, program mencetak data dari setiap node, dan menggerakkan pointer `last` ke node sebelumnya dengan menggunakan pointer `prev`. Proses ini memungkinkan pencetakan isi linked list baik secara maju maupun mundur dengan menggunakan pendekatan yang efisien dan

			sederhana.

	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;
		}
			Bagian program ini adalah sebuah fungsi utama yang menguji fungsi-fungsi yang telah diimplementasikan sebelumnya dalam linked list ganda (doubly linked list). Pertama, pointer `head` yang menunjuk ke node pertama dalam linked list dideklarasikan dan diinisialisasi dengan NULL, menandakan linked list kosong. Selanjutnya, tiga node baru ditambahkan ke linked list menggunakan fungsi `push`, dengan nilai-nilai 6, 5, dan 2 secara berurutan. Kemudian, sebuah node baru disisipkan setelah node kedua dalam linked list menggunakan fungsi `insertAfter`, dengan nilai 5. Program mencetak isi linked list baik secara maju maupun mundur. Setelah mencetak isi linked list, program menunggu input dari pengguna sebelum mengakhiri eksekusi.

3	5-10	struct Node
		{
		int data;
		struct Node *next; // Pointer to next node
		struct Node *prev; // Pointer to previous node
		};
			Bagian ini adalah definisi dari struktur data `Node` yang digunakan untuk membuat doubly linked list. Setiap node dalam struktur ini memiliki tiga komponen utama: `int data` untuk menyimpan nilai data, `struct Node *next` untuk menunjuk ke node berikutnya, dan `struct Node *prev` untuk menunjuk ke node sebelumnya. Dengan struktur ini, kita dapat membuat dan mengelola linked list yang memungkinkan untuk traversal maju dan mundur dengan mudah.

	11-25	void push(Node** head_ref, int new_data)
		{
		/* 1. allocate node */
		Node* new_node = new Node();
		/* 2. put in the data */
		new_node->data = new_data;
		/* 3. Make next of new node as head and previous as NULL */
		new_node->next = (*head_ref);
		new_node->prev = NULL;
			Fungsi `push` untuk menambahkan node baru ke depan linked list, terutama digunakan dalam doubly linked list. Pertama, memori dialokasikan untuk node baru menggunakan `new`. Selanjutnya, data node diisi dengan nilai yang diberikan. Pointer `next` dari node baru diatur menunjuk ke kepala saat ini, dan pointer `prev` diatur sebagai `NULL`. Jika linked list tidak kosong, pointer `prev` dari kepala sebelumnya

		/* 4. change prev of head node to new node */
		if ((*head_ref) != NULL)
		(*head_ref)->prev = new_node;
		/* 5. move the head to point to the new node */
		(*head_ref) = new_node;
		}
			diubah untuk menunjuk ke node baru. Akhirnya, pointer kepala diperbarui untuk menunjuk ke node baru, menjadikannya sebagai kepala baru linked list. Dengan ini, fungsi `push` memungkinkan penambahan node baru ke depan linked list dengan efisiensi.

	26-51	void append(struct Node** head_ref, int new_data)
		{
		/* 1. allocate node */
		struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
		struct Node* last = *head_ref; /* used in step 5*/
		/* 2. put in the data */
		new_node->data = new_data;
		/* 3. This new node is going to be the last node, so
		make next of it as NULL*/
		new_node->next = NULL;
		/* 4. If the Linked List is empty, then make the new
		node as head */
		if (*head_ref == NULL) {
		new_node->prev = NULL;
		*head_ref = new_node;
		return;
		}
		/* 5. Else traverse till the last node */
		while (last->next != NULL)
		last = last->next;
		/* 6. Change the next of last node */
		last->next = new_node;
		/* 7. Make last node as previous of new node */
		new_node->prev = last;
		return;
		}
			Fungsi `append` dalam program bertujuan menambahkan node baru ke akhir linked list, terutama dalam konteks doubly linked list. Pertama, dilakukan alokasi memori untuk node baru. Data node diatur dengan nilai yang diberikan, dan pointer `next` diatur menjadi `NULL`, menandakan node baru akan menjadi node terakhir. Jika linked list kosong, node baru menjadi kepala. Jika tidak, dilakukan traversal hingga akhir linked list untuk menambahkan node baru di belakangnya. Dengan menyesuaikan pointer `prev` dari node baru, node baru dihubungkan ke node terakhir sebelumnya. Dengan demikian, fungsi `append` memungkinkan penambahan node baru ke akhir linked list dalam doubly linked list.

	52-66	void printList(struct Node* node) {	Bagian program tersebut merupakan implementasi dari fungsi `printList`

		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;
		}
		}
			untuk mencetak isi dari linked list. Fungsi ini menerima pointer ke node pertama dari linked list sebagai parameter. Pertama, fungsi mencetak isi linked list dari awal ke akhir dengan menggunakan loop while, kemudian mencetak isi dari linked list dari akhir ke awal dengan menggunakan pointer `last` yang menunjuk ke node terakhir dan mengikuti pointer `prev` dari node ke node sebelumnya hingga mencapai NULL. Dengan demikian, fungsi ini mencetak isi dari linked list dalam dua arah, dari awal ke akhir dan dari akhir ke awal.

	67-86	int main()
		{
		/* Start with the empty list */
		struct Node* head = NULL;
		// Insert 6. So linked list becomes 6->NULL
		append(&head, 6);
		// Insert 7 at the beginning. So
		// linked list becomes 7->6->NULL
		push(&head, 7);
		// Insert 1 at the beginning. So
		// linked list becomes 1->7->6->NULL
		push(&head, 1);
		// Insert 4 at the end. So linked
		// list becomes 1->7->6->4->NULL
		append(&head, 4);
		printf("Created DLL is: ");
		printList(head);
		getchar();
		return 0;
		}
			Fungsi `main`bertindak sebagai program utama. Pada awalnya, sebuah pointer `head` dideklarasikan dan diinisialisasi dengan NULL, menandakan bahwa linked list awalnya kosong. Selanjutnya, dilakukan serangkaian pemanggilan fungsi untuk menambahkan node-node baru ke linked list. Pertama, dilakukan pemanggilan fungsi `append(&head, 6);` untuk menambahkan node dengan nilai 6 ke akhir linked list. Kemudian, fungsi `push(&head, 7);` dipanggil untuk menambahkan node dengan nilai 7 di awal linked list, diikuti dengan pemanggilan `push(&head, 1);` untuk menambahkan node dengan nilai 1 juga di awal linked list. Terakhir, pemanggilan `append(&head, 4);` dilakukan untuk menambahkan node dengan nilai 4 ke akhir linked list. Setelah itu, pesan "Created DLL is: " dicetak ke layar, yang diikuti oleh pemanggilan `printList(head);` untuk mencetak isi dari linked list ke layar, dimulai dari node yang ditunjuk oleh `head`. Program menunggu hingga pengguna menekan tombol Enter sebelum akhirnya mengembalikan

			nilai 0, menandakan bahwa program telah berakhir dengan sukses.

4	2-6	struct Node {
		int data;
		struct Node* next;
		struct Node* prev;
		};
			Bagian program tersebut mendefinisikan sebuah struktur bernama `Node` yang merepresentasikan simpul dalam linked list ganda. Setiap simpul memiliki tiga bagian: `data` untuk menyimpan nilai data, `next` untuk menunjuk ke simpul berikutnya, dan `prev` untuk menunjuk ke simpul sebelumnya. Dengan struktur ini, operasi pada linked list seperti penambahan, penghapusan, dan penelusuran dapat dilakukan dengan efisien.

	7-16	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 push yang bertujuan untuk menambahkan sebuah node baru ke awal linked list ganda. Pertama-tama, memori dialokasikan untuk node baru menggunakan fungsi malloc, dan nilai data baru yang diterima sebagai parameter dimasukkan ke dalam node tersebut. Selanjutnya, pointer next dari node baru diatur untuk menunjuk ke node yang sebelumnya menjadi kepala linked list. Karena node baru akan menjadi node pertama dalam linked list, pointer prev-nya diatur menjadi NULL. Namun, jika linked list tidak kosong, pointer prev dari node yang sebelumnya menjadi kepala linked list diatur untuk menunjuk ke node baru. Terakhir, pointer head_ref yang merepresentasikan kepala linked list diperbaharui untuk menunjuk ke node baru, sehingga node baru tersebut menjadi kepala baru dari linked list. Dengan demikian, fungsi push ini berhasil menambahkan node baru ke awal linked list ganda, dan menjaga keterhubungan linked list dengan baik.

	8-42	void insertBefore(struct Node** head_ref, struct Node* next_node, int new_data) { /*1. check if the given	Fungsi `insertBefore` bertujuan untuk menyisipkan sebuah node baru sebelum node yang ditentukan dalam linked list ganda. Langkah-langkah dalam fungsi ini adalah sebagai

		next_node is NULL */
		if (next_node == NULL) {
		printf("the given next node cannot be NULL");
		return;
		}
		/* 2. allocate new node */
		struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
		/* 3. put in the data */
		new_node->data = new_data;
		/* 4. Make prev of new node as prev of next_node */
		new_node->prev = next_node->prev;
		/* 5. Make the prev of next_node as new_node */
		next_node->prev = new_node;
		/* 6. Make next_node as next of new_node */
		new_node->next = next_node;
		/* 7. Change next of new_node's previous node */
		if (new_node->prev != NULL)
		new_node->prev->next = new_node;
		/* 8. If the prev of new_node is NULL, it will be
		the new head node */
		else
		(*head_ref) = new_node;
		}
			berikut: Pertama, fungsi melakukan pengecekan apakah node berikutnya (`next_node`) yang diberikan tidak bernilai NULL. Jika ternyata `next_node` adalah NULL, maka fungsi akan menampilkan pesan kesalahan yang menyatakan bahwa node berikutnya tidak boleh NULL, dan proses akan dihentikan. Selanjutnya, fungsi mengalokasikan memori untuk node baru menggunakan fungsi `malloc`, kemudian memasukkan nilai data baru (`new_data`) ke dalam node tersebut. Pointer `prev` dari node baru diatur untuk menunjuk ke node sebelumnya dari `next_node`, sehingga mempertahankan keterhubungan dengan node sebelumnya. Pointer `prev` dari `next_node` diubah untuk menunjuk ke node baru, sehingga node baru disisipkan di antara node sebelumnya dan `next_node`. Pointer `next` dari node baru diatur untuk menunjuk ke `next_node`, sehingga node baru terhubung dengan node berikutnya. Jika node sebelumnya ada, pointer `next` dari node sebelumnya diubah untuk menunjuk ke node baru. Namun, jika tidak ada node sebelumnya (artinya, node baru akan menjadi kepala baru), maka pointer `head_ref` yang merepresentasikan kepala linked list diperbaharui untuk menunjuk ke node baru. Dengan langkah-langkah ini, fungsi `insertBefore` dapat digunakan untuk menyisipkan node baru sebelum node tertentu dalam linked list ganda dan menjaga keterhubungan linked list dengan baik.

	43-57	void printList(struct Node* node)
		{
		struct Node* last;
		printf("\nTraversal in forward direction \n");
		while (node != NULL) {
		printf(" %d ", node-
			Fungsi `printList` digunakan untuk mencetak isi dari linked list ganda. Pertama-tama, variabel `last` dideklarasikan sebagai pointer ke node terakhir dalam linked list. Selanjutnya, fungsi ini melakukan traversal dari awal ke akhir linked list

		>data);
		last = node;
		node = node->next;
		}
		printf("\nTraversal in reverse direction \n");
		while (last != NULL) {
		printf(" %d ", last->data);
		last = last->prev;
		}
		}
			dengan looping menggunakan pointer `node`, mencetak nilai `data` dari setiap node, dan memperbarui pointer `last`. Setelah itu, pesan ditampilkan untuk menandakan penelusuran dari akhir ke awal, dan dilakukan penelusuran mundur dengan looping menggunakan pointer `last`, mencetak nilai `data` dari setiap node, dan memperbarui `last` ke node sebelumnya. Dengan demikian, fungsi ini mencetak isi linked list ganda baik dari awal ke akhir maupun dari akhir ke awal.

	60-72	int main()
		{
		/* Start with the empty list */
		struct Node* head = NULL;
		push(&head, 7);
		push(&head, 1);
		push(&head, 4);
		insertBefore(&head, head->next, 8);
		printf("Created DLL is: ");
		printList(head);
		getchar();
		return 0;
		}
			Fungsi main(), yang merupakan titik awal dari eksekusi. Di dalam main(), sebuah DLL kosong dideklarasikan dengan node awal (head) diatur menjadi NULL. Selanjutnya, dilakukan pemanggilan fungsi push() sebanyak tiga kali untuk menambahkan node-node baru dengan nilai 7, 1, dan 4 secara berurutan di awal DLL. Kemudian, dilakukan pemanggilan fungsi insertBefore() untuk menyisipkan node baru dengan nilai 8 sebelum node kedua dalam DLL. Setelah operasi-operasi tersebut selesai dilakukan, isi dari DLL dicetak menggunakan fungsi printList(). Penggunaan getchar() digunakan untuk menahan konsol agar tidak langsung tertutup, memberikan waktu bagi pengguna untuk melihat hasil sebelum program selesai dieksekusi. Program kemudian mengembalikan nilai 0 dari fungsi main() untuk menandakan bahwa program berakhir dengan sukses. Dengan demikian, program ini digunakan untuk menguji dan memahami operasi dasar pada Doubly Linked List dalam bahasa pemrograman C.