SOAL 1 Buat sebuah fungsi yang menerima string dan mengembalikannya dalam keadaan terbalik. Anda harus menggunakan struktur data Stack yang diimplementasikan secara manual menggunakan singly linked list. Struktur Data: ```c++ #include #include using namespace std; struct Node { char data; Node* next; }; ``` Lengkapi Fungsi berikut: ```c++ // Fungsi Push dan Pop manual bisa dibuat di sini atau di dalam fungsi utama. // Disarankan untuk membuatnya terpisah agar lebih rapi. void push(Node*& top, char data) { Node* newNode = new Node{data, top}; top = newNode; } char pop(Node*& top) { if (top == nullptr) return '\0'; // Return null character jika stack kosong Node* temp = top; char poppedValue = temp->data; top = top->next; delete temp; return poppedValue; } string reverseString(string s) { Node* stackTop = nullptr; string reversed = ""; // --- LENGKAPI DI SINI --- // 1. Push setiap karakter dari string s ke dalam stack. // 2. Pop setiap karakter dari stack dan tambahkan ke string `reversed`. // --- LENGKAPI DI SINI --- return reversed; } int main() { string text = "Struktur Data"; cout << "Teks asli: " << text << endl; cout << "Teks terbalik: " << reverseString(text) << endl; // Output: ataD rukurts return 0; } ``` Hanya Info: Karena sifat Stack adalah LIFO (Last-In, First-Out), karakter terakhir yang dimasukkan akan menjadi yang pertama kali dikeluarkan. Ini secara alami membalik urutan karakter. --- ## SOAL 2 Buat fungsi yang memeriksa apakah sebuah string yang berisi tanda kurung (), {}, dan [] seimbang. Contohnya, "{[()]}" seimbang, tetapi "{[(])}" tidak. Gunakan implementasi Stack manual. Struktur Data: ```c++ #include #include using namespace std; struct Node { char data; Node* next; }; ``` Lengkapi Fungsi berikut: ```c++ // Anda bisa menggunakan fungsi push dan pop dari soal sebelumnya. bool areBracketsBalanced(string expr) { Node* stackTop = nullptr; // --- LENGKAPI DI SINI --- // 1. Loop setiap karakter dalam `expr`. // 2. Jika karakter adalah kurung buka '(', '{', '[', push ke stack. // 3. Jika karakter adalah kurung tutup ')', '}', ']', cek: // a. Apakah stack kosong? Jika ya, return false. // b. Pop stack, lalu cek apakah kurung tutup cocok dengan kurung buka. Jika tidak, return false. // 4. Setelah loop selesai, jika stack kosong, return true. Jika tidak, return false. // --- LENGKAPI DI SINI --- return false; // Placeholder } int main() { string expr1 = "{[()]}"; cout << expr1 << " -> " << (areBracketsBalanced(expr1) ? "Seimbang" : "Tidak Seimbang") << endl; // Expected output: Seimbang string expr2 = "{[(])}"; cout << expr2 << " -> " << (areBracketsBalanced(expr2) ? "Seimbang" : "Tidak Seimbang") << endl; return 0; } ``` Hanya Info: Stack digunakan untuk menyimpan kurung buka. Setiap kali menemukan kurung tutup, kita memeriksa apakah kurung buka terakhir di stack adalah pasangannya. --- ## SOAL 3 Buat implementasi Queue manual menggunakan linked list untuk simulasi antrian printer yang memproses dokumen berdasarkan urutan kedatangan (FIFO). Struktur Data: ```c++ #include #include using namespace std; struct Node { string document; Node* next; }; ``` Lengkapi Fungsi berikut: ```c++ void enqueue(Node*& front, Node*& rear, string document) { Node* newNode = new Node{document, nullptr}; // --- LENGKAPI DI SINI --- // 1. Jika queue kosong (front == nullptr), set front dan rear ke newNode // 2. Jika tidak kosong, sambungkan rear->next ke newNode, lalu update rear // --- LENGKAPI DI SINI --- } string dequeue(Node*& front, Node*& rear) { if (front == nullptr) return ""; // Queue kosong // --- LENGKAPI DI SINI --- // 1. Simpan data dari front node // 2. Geser front ke front->next // 3. Jika front menjadi nullptr, set rear juga ke nullptr // 4. Delete node lama dan return data // --- LENGKAPI DI SINI --- return ""; // Placeholder } void processAllDocuments(Node*& front, Node*& rear) { // --- LENGKAPI DI SINI --- // Loop hingga queue kosong, dequeue dan print setiap dokumen // Format: "Memproses: [nama_dokumen]" // --- LENGKAPI DI SINI --- } int main() { Node* front = nullptr; Node* rear = nullptr; enqueue(front, rear, "Document1.pdf"); enqueue(front, rear, "Report.docx"); enqueue(front, rear, "Presentation.pptx"); cout << "Memulai pemrosesan antrian printer:" << endl; processAllDocuments(front, rear); return 0; } ``` Hanya Info: Queue menggunakan prinsip FIFO (First-In, First-Out). Dokumen yang pertama masuk antrian akan diproses pertama kali. Pointer front menunjuk ke elemen yang akan dikeluarkan, dan rear menunjuk ke posisi untuk elemen baru. Output: ```c++ Memulai pemrosesan antrian printer: Memproses: Document1.pdf Memproses: Report.docx Memproses: Presentation.pptx ``` --- ## SOAL 4 Anda memiliki sebuah circular doubly linked list yang datanya sudah terurut (ascending). Buatlah fungsi untuk menyisipkan sebuah node baru ke dalam list tersebut sehingga urutannya tetap terjaga. Struktur Data: ```c++ #include using namespace std; struct Node { int data; Node* next; Node* prev; }; ``` Lengkapi Fungsi berikut: ```c++ /* * Fungsi ini menerima referensi ke pointer head dan data yang akan disisipkan. * Pointer head bisa berubah jika data baru menjadi elemen terkecil. */ void sortedInsert(Node *&head_ref, int data) { Node*

newNode = new Node{data, nullptr, nullptr}; // Kasus 1: List masih kosong if (head_ref == nullptr) { newNode->next = newNode; newNode->prev = newNode; head_ref = newNode; return; } // --- LENGKAPI DI SINI --- // Kasus 2: Data baru lebih kecil dari head (sisipkan di awal) // 1. Jika data < head_ref->data, sisipkan sebelum head dan update head_ref // Kasus 3: Cari posisi yang tepat (tengah/akhir) // 1. Gunakan pointer current mulai dari head_ref // 2. Loop: while (current->next != head_ref && current->next->data < data) // 3. Setelah loop, sisipkan newNode setelah current // 4. Pastikan update semua pointer next dan prev dengan benar // // CATATAN: Jika data sama dengan existing data, sisipkan setelahnya // --- LENGKAPI DI SINI --- } void printList(Node *head_ref) { if (head_ref == nullptr) { cout << "List kosong" << endl; return; } Node *current = head_ref; do { cout << current->data << " "; current = current->next; } while (current != head_ref); cout << endl; } int main() { Node *head = nullptr; // Test sorted insert sortedInsert(head, 30); sortedInsert(head, 10); sortedInsert(head, 40); sortedInsert(head, 20); cout << "Circular Doubly Linked List (sorted): "; // Expected output: 10 20 30 40 printList(head); return 0; } ``` Hanya Info: Karena listnya sirkular dan terurut, kita bisa dengan cepat memeriksa apakah node baru harus diletakkan di awal (lebih kecil dari head) atau di akhir (lebih besar dari tail/head->prev). Jika tidak, kita baru melakukan traversal untuk mencari posisi di tengah. --- ## SOAL 5 Buat sebuah fungsi untuk menukar posisi node head dan node tail dalam sebuah circular doubly linked list tanpa menukar datanya, melainkan dengan memanipulasi pointernya. Struktur Data: ```c++ #include using namespace std; struct Node { int data; Node* next; Node* prev; }; ``` Lengkapi Fungsi berikut: ```c++ /* * Fungsi ini menerima referensi ke pointer head dan tail. * Pointer head dan tail akan di-update setelah penukaran. */ void exchangeHeadAndTail(Node *&head_ref) { // Hanya berjalan jika ada 2 node atau lebih if (head_ref == nullptr || head_ref->next == head_ref) { return; } Node* head = head_ref; Node* tail = head_ref->prev; // Tail adalah prev dari head // Hal yang perlu dilakukan: // Buat kondisi jika hanya 2 node, cukup swap head_ref // Simpan neighbor (yaitu head_next dan tail_prev) // Update koneksi: tail_prev <-> tail <-> head_next // Update koneksi: head_next <- ... -> tail_prev <-> head <-> tail_prev // terakhir Update head_ref // kondisi bisa kalian sesuaikan sendiri tapi usahakan outputnya sama } void printList(Node *head_ref) { if (head_ref == nullptr) { cout << "List kosong" << endl; return; } Node *current = head_ref; do { cout << current->data << " "; current = current->next; } while (current != head_ref); cout << endl; } void insertEnd(Node *&head_ref, int data) { Node *newNode = new Node{data, nullptr, nullptr}; if (head_ref == nullptr) { newNode->next = newNode; newNode->prev = newNode; head_ref = newNode; return; } Node *tail = head_ref->prev; newNode->next = head_ref; newNode->prev = tail; head_ref->prev = newNode; tail->next = newNode; } int main() { Node *head = nullptr; // Buat list: 1 <-> 2 <-> 3 <-> 4 <-> 5 insertEnd(head, 1); insertEnd(head, 2); insertEnd(head, 3); insertEnd(head, 4); insertEnd(head, 5); cout << "List sebelum exchange: "; printList(head); exchangeHeadAndTail(head); cout << "List setelah exchange head dan tail: "; // Expected output: 5 2 3 4 1 printList(head); return 0; } ``` Hanya Info: Kunci untuk soal ini adalah menggambar dan memvisualisasikan koneksi antar node. Dengan menyimpan node tetangga (head_next dan tail_prev) sebelum melakukan perubahan, kita memiliki semua referensi yang dibutuhkan untuk menyambungkan kembali list dalam urutan yang baru tanpa kehilangan node manapun.