Nama : M Alief Alfaridzi

Kelas : IF 03-03

NIM : 1203230045

Tugas OTH Circular Double Linked List

Source Code

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = newNode->prev = newNode;
    return newNode;
}

void append(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* last = (*head)->prev;
        newNode->next = *head;
        (*head)->prev = newNode;
        newNode->prev = last;
        last->next = newNode;
    }
}

void printList(Node* head) {
    if (head == NULL) return;
    Node* temp = head;
    do {
        printf("Address: %p, Data: %d\n", (void*)temp, temp->data);
        temp = temp->next;
    } while (temp != head);
}

void sortList(Node** head) {
    if (*head == NULL) return;
```

```
    Node* current = *head;
    Node* index = NULL;
    Node* temp = NULL;

    do {
        index = current->next;
        while (index != *head) {
            if (current->data > index->data) {
                Node* prevCurrent = current->prev;
                Node* nextCurrent = current->next;
                Node* prevIndex = index->prev;
                Node* nextIndex = index->next;

                if (current->next == index) {
                    current->next = nextIndex;
                    current->prev = index;
                    index->next = current;
                    index->prev = prevCurrent;
                    if (nextIndex != *head) nextIndex->prev = current;
                    if (prevCurrent != *head) prevCurrent->next = index;
                } else {
                    current->next = nextIndex;
                    current->prev = prevIndex;
                    index->next = nextCurrent;
                    index->prev = prevCurrent;
                    if (nextIndex != *head) nextIndex->prev = current;
                    if (prevIndex != *head) prevIndex->next = current;
                    if (nextCurrent != *head) nextCurrent->prev = index;
                    if (prevCurrent != *head) prevCurrent->next = index;
                }

                if (*head == current) *head = index;
                else if (*head == index) *head = current;

                temp = current;
                current = index;
                index = temp;
            }
            index = index->next;
        }
        current = current->next;
    } while (current->next != *head);
}

int main() {
    Node* head = NULL;
    int N, data;
```

```
    printf("Masukkan jumlah data: ");
    scanf("%d", &N);

    for (int i = 0; i < N; i++) {
        printf("Masukkan data ke-%d: ", i + 1);
        scanf("%d", &data);
        append(&head, data);
    }

    printf("List sebelum pengurutan:\n");
    printList(head);

    sortList(&head);

    printf("List setelah pengurutan:\n");
    printList(head);

    return 0;
}
```

Penjelasan Code Program

1. #include <stdio.h>
    - Baris ini adalah preprocessor command yang menginstruksikan compiler untuk memasukkan standard input-output library ke dalam program. Library ini digunakan untuk operasi input dan output, seperti fungsi printf atau scanf.
2. #include <stdlib.h>
    - Baris ini menginstruksikan compiler untuk memasukkan standard library, yang berisi fungsi-fungsi penting seperti malloc, free, dan lain-lain. Fungsi malloc digunakan untuk alokasi memori dalam program
3. typedef struct Node {
    - typedef digunakan untuk mendefinisikan tipe baru. Di sini, struct Node didefinisikan sebagai tipe baru dengan nama Node. Ini memudahkan deklarasi variabel dari tipe ini di kemudian hari
4. int data;
    - Ini mendeklarasikan variabel data sebagai tipe int di dalam struktur Node. Variabel ini akan digunakan untuk menyimpan data integer dalam node.
5. struct Node* next;

- Mendeklarasikan pointer next yang menunjuk ke struktur Node berikutnya. Ini digunakan untuk menautkan node saat ini ke node berikutnya dalam linked list.

6. struct Node* prev;
   - Sama seperti next, tetapi prev menunjuk ke node sebelumnya, membuat ini menjadi struktur doubly linked list, di mana setiap node terhubung ke node sebelum dan sesudahnya.

7. } Node;
   - Menutup definisi struct Node dan mengakhiri typedef dengan nama Node.

8. Node* createNode(int data) {
   - Mendefinisikan fungsi createNode yang mengambil satu parameter integer data dan mengembalikan pointer ke Node. Fungsi ini digunakan untuk membuat node baru.

9. Node* newNode = (Node*)malloc(sizeof(Node));
   - Mengalokasikan memori untuk node baru menggunakan malloc yang mengembalikan pointer ke memori yang dialokasikan. Ukuran memori yang dialokasikan adalah ukuran dari Node. Pointer tersebut di-cast ke (Node*) untuk memastikan tipe data yang benar.

10. newNode->data = data;
    - Mengatur nilai data dari newNode dengan nilai yang diberikan melalui parameter fungsi.

11. newNode->next = newNode->prev = newNode;
    - Menginisialisasi pointer next dan prev dari newNode untuk menunjuk ke dirinya sendiri. Ini biasanya dilakukan saat node baru adalah satu-satunya node dalam linked list, sehingga ia menunjuk ke dirinya sendiri sebagai node berikutnya dan sebelumnya.

12. return newNode;
    - Mengembalikan pointer ke node baru yang telah dibuat dan diinisialisasi.

13. void append(Node** head, int data) {

- Mendefinisikan fungsi append yang menerima pointer ke pointer dari head (yang menunjuk ke kepala list) dan sebuah integer data yang akan disimpan dalam node baru.

14. Node* newNode = createNode(data);
    - Membuat node baru dengan memanggil fungsi createNode dan menyimpan hasilnya dalam pointer newNode.
15. if (*head == NULL) {
    - Memeriksa apakah list kosong (yaitu, head menunjuk ke NULL). Jika ya, maka ini adalah penambahan node pertama ke dalam list.
16. *head = newNode;
    - Menetapkan newNode sebagai node pertama dan satu-satunya dalam list, sehingga head sekarang menunjuk ke newNode.
17. } else {
    - Jika list tidak kosong, eksekusi akan melanjutkan ke bagian ini untuk menambahkan newNode ke akhir list.
18. Node* last = (*head)->prev;
    - Mengambil node terakhir dari list, yang dapat diakses melalui pointer prev dari head.
19. newNode->next = *head;
    - Menetapkan pointer next dari newNode untuk menunjuk kembali ke head, sehingga membentuk lingkaran.
20. (*head)->prev = newNode;
    - Menetapkan pointer prev dari head untuk menunjuk ke newNode, menghubungkan head dengan node baru di belakangnya.
21. newNode->prev = last;
    - Menetapkan pointer prev dari newNode untuk menunjuk ke node terakhir yang ada sebelumnya.
22. last->next = newNode;
    - Menetapkan pointer next dari node terakhir untuk menunjuk ke newNode, menyelesaikan sambungan dua arah.
23. void printList(Node* head) {
    - Mendefinisikan fungsi printList yang menerima pointer head ke node pertama dalam list.
24. if (head == NULL) return;

- Jika head adalah NULL, artinya list kosong, fungsi akan langsung kembali tanpa melakukan apa-apa.

25. Node* temp = head;
    - Mendeklarasikan pointer temp dan menginisialisasinya dengan head untuk mulai iterasi dari node pertama.
26. do {
    - Memulai loop do-while yang akan berjalan setidaknya sekali, memungkinkan pencetakan bahkan jika list hanya memiliki satu node.
27. printf("Address: %p, Data: %d\n", (void*)temp, temp->data);
    - Mencetak alamat memori dan data dari node yang ditunjuk oleh temp.
28. temp = temp->next;
    - Memindahkan temp ke node berikutnya dalam list.
29. } while (temp != head);
    - Loop akan terus berjalan sampai temp kembali ke head, menandakan bahwa semua node telah dicetak.
30. void sortList(Node** head) {
    - Mendefinisikan fungsi sortList yang menerima pointer ke pointer head (yang menunjuk ke kepala list) sebagai argumen.
31. if (*head == NULL) return;
    - Memeriksa apakah list kosong (yaitu, head menunjuk ke NULL). Jika ya, fungsi langsung kembali tanpa melakukan apa-apa.
32. Node* current = *head;
    - Membuat pointer current dan menginisialisasinya dengan head untuk memulai iterasi dari node pertama dalam list.
33. Node* index = NULL;
    - Membuat pointer index dan menginisialisasinya dengan NULL. Pointer ini akan digunakan untuk membandingkan nilai data antara node saat ini (current) dan node berikutnya dalam list.
34. Node* temp = NULL;
    - Membuat pointer temp dan menginisialisasinya dengan NULL. Pointer ini akan digunakan untuk pertukaran node saat melakukan pengurutan.
35. do {

- Memulai loop do-while untuk mengulangi proses pengurutan setidaknya sekali. Loop ini akan berjalan hingga current kembali ke head, menunjukkan bahwa semua elemen telah diurutkan.

36. index = current->next;
    - Mengatur index ke node berikutnya setelah current. Ini akan digunakan untuk membandingkan nilai data antara current dan index.
37. while (index != *head) {
    - Memulai loop while yang berjalan selama index tidak sama dengan head. Loop ini akan membandingkan nilai data antara current dan index untuk menentukan apakah perlu dilakukan pertukaran.
38. if (current->data > index->data) {
    - Memeriksa apakah nilai data dari current lebih besar dari nilai data dari index. Jika ya, berarti perlu dilakukan pertukaran.
39. Node* prevCurrent = current->prev;
    - Membuat pointer prevCurrent dan menginisialisasinya dengan node sebelumnya dari current.
40. Node* nextCurrent = current->next;
    - Membuat pointer nextCurrent dan menginisialisasinya dengan node setelah current.
41. Node* prevIndex = index->prev;
    - Membuat pointer prevIndex dan menginisialisasinya dengan node sebelumnya dari index.
42. Node* nextIndex = index->next;
    - Membuat pointer nextIndex dan menginisialisasinya dengan node setelah index.
43. if (current->next == index) {
    - Memeriksa apakah current dan index bersebelahan (node current berada sebelum node index).
44. current->next = nextIndex;
    - Mengatur pointer next dari current untuk menunjuk ke nextIndex.
45. current->prev = index;
    - Mengatur pointer prev dari current untuk menunjuk ke index.
46. index->next = current;
    - Mengatur pointer next dari index untuk menunjuk ke current.

47. index->prev = prevCurrent;
    - Mengatur pointer prev dari index untuk menunjuk ke prevCurrent.
48. if (nextIndex != *head) nextIndex->prev = current;
    - Memeriksa apakah nextIndex bukan head. Jika iya, mengatur pointer prev dari nextIndex untuk menunjuk ke current.
49. if (prevCurrent != *head) prevCurrent->next = index;
    - Memeriksa apakah prevCurrent bukan head. Jika iya, mengatur pointer next dari prevCurrent untuk menunjuk ke index.
50. } else {
    - Jika current dan index tidak bersebelahan (berjarak lebih dari satu node), eksekusi akan melanjutkan ke bagian ini.
51. current->next = nextIndex;
    - Mengatur pointer next dari current untuk menunjuk ke nextIndex.
52. current->prev = prevIndex;
    - Mengatur pointer prev dari current untuk menunjuk ke prevIndex.
53. index->next = nextCurrent;
    - Mengatur pointer next dari index untuk menunjuk ke nextCurrent.
54. index->prev = prevCurrent;
    - Mengatur pointer prev dari index untuk menunjuk ke prevCurrent.
55. if (nextIndex != *head) nextIndex->prev = current;
    - Memeriksa apakah nextIndex bukan head. Jika iya, mengatur pointer prev dari nextIndex untuk menunjuk ke current.
56. if (prevIndex != *head) prevIndex->next = current;
    - Memeriksa apakah prevIndex bukan head. Jika iya, mengatur pointer next dari prevIndex untuk menunjuk ke current.
57. if (nextCurrent != *head) nextCurrent->prev = index;
    - Memeriksa apakah nextCurrent bukan head. Jika iya, mengatur pointer prev dari nextCurrent untuk menunjuk ke index.
58. if (prevCurrent != *head) prevCurrent->next = index;
    - Memeriksa apakah prevCurrent bukan head. Jika iya, mengatur pointer next dari prevCurrent untuk menunjuk ke index.
59. if (*head == current) *head = index;
    - Memeriksa apakah current adalah head. Jika iya, mengatur head untuk menunjuk ke index.

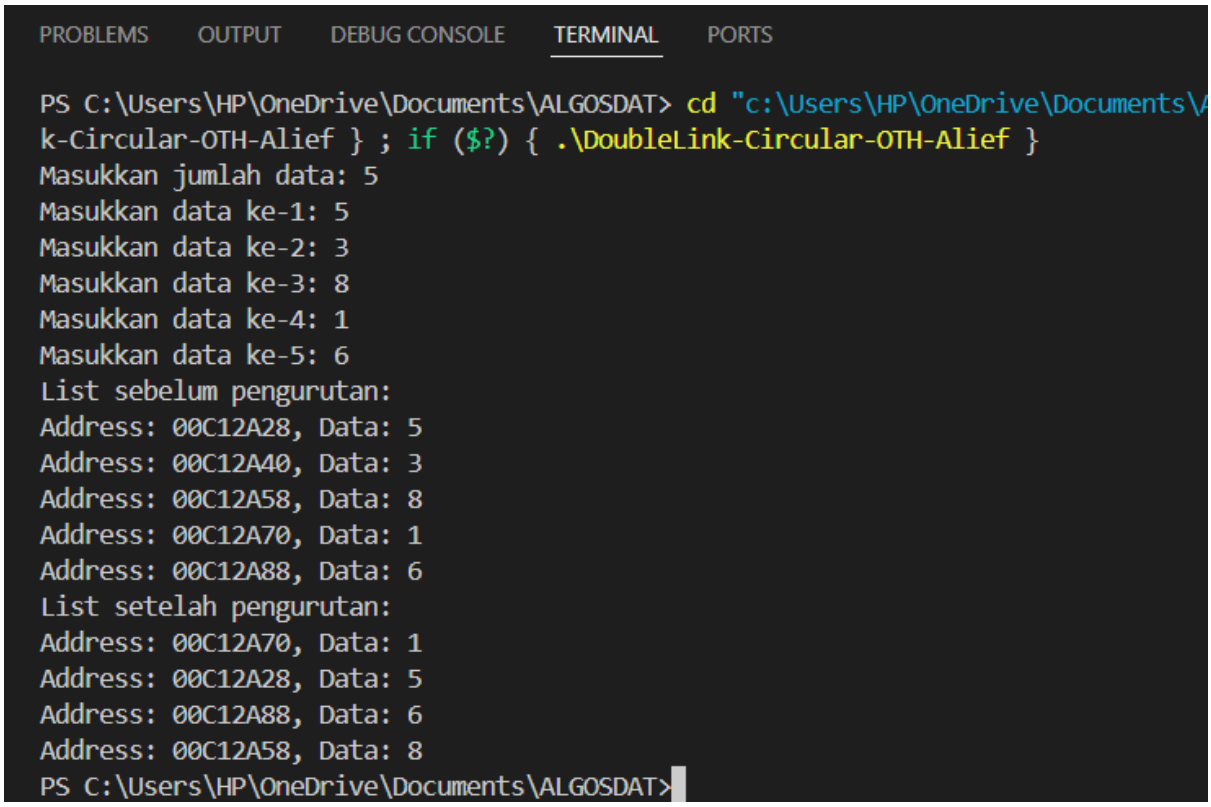
60. else if (*head == index) *head = current;
    - Memeriksa apakah index adalah head. Jika iya, mengatur head untuk menunjuk ke current.
61. temp = current;
    - Menukar nilai current dan index dengan menggunakan variabel temp sebagai tempat penyimpanan sementara.
62. current = index;
    - Mengatur current menjadi index untuk melanjutkan iterasi ke node berikutnya.
63. index = temp;
    - Mengatur index menjadi temp untuk melanjutkan iterasi ke node berikutnya.
64. } while (current->next != *head);
    - Loop akan terus berjalan selama current memiliki node berikutnya yang bukan head, menunjukkan bahwa pengurutan belum selesai.
65. Int main() {
    - Mendeklarasikan fungsi masuk main utama program.
66. Node* head = NULL;
    - Mendeklarasikan pointer head yang menunjuk ke NULL sebagai awal dari circular doubly linked list.
67. int N, data;
    - Mendeklarasikan variabel N dan data yang akan digunakan untuk menyimpan jumlah data yang akan dimasukkan dan nilai data yang dimasukkan oleh pengguna.
68. printf("Masukkan jumlah data: ");
    - Mencetak pesan untuk meminta pengguna memasukkan jumlah data yang akan dimasukkan ke dalam list.
69. scanf("%d", &N);
    - Mengambil input jumlah data yang dimasukkan oleh pengguna dan menyimpannya dalam variabel N.
70. for (int i = 0; i < N; i++) {
    - Memulai loop for untuk meminta pengguna memasukkan data sebanyak N kali.
71. printf("Masukkan data ke-%d: ", i + 1);
    - Mencetak pesan untuk meminta pengguna memasukkan data ke-i+1.

72. scanf("%d", &data);
    - Mengambil input data yang dimasukkan oleh pengguna dan menyimpannya dalam variabel data.
73. append(&head, data);
    - Memanggil fungsi append untuk menambahkan data yang dimasukkan oleh pengguna ke dalam list.
74. printf("List sebelum pengurutan:\n");
    - Mencetak pesan untuk menampilkan list sebelum dilakukan pengurutan.
75. printList(head);
    - Memanggil fungsi printList untuk mencetak isi dari list sebelum dilakukan pengurutan.
76. sortList(&head);
    - Memanggil fungsi sortList untuk mengurutkan isi dari list.
77. printf("List setelah pengurutan:\n");
    - Mencetak pesan untuk menampilkan list setelah dilakukan pengurutan.
78. printList(head);
    - Memanggil fungsi printList untuk mencetak isi dari list setelah dilakukan pengurutan.
79. return 0;
    - Mengembalikan nilai 0, menandakan bahwa program telah berakhir dengan sukses.

Output Program

Pada output di bawah, list sebelum pengurutan mencetak alamat memori dan nilai data dari setiap node dalam list. Setelah pengurutan, list dicetak kembali dengan nilai data yang sudah diurutkan secara ascending. Dan juga memory address dan data tidak ada yang dimanipulasi sesuai dengan address dan data yang pertama keluar sebelum diurutkan.

Contoh Output 1



Contoh Output 2