Laporan OTH

Circular Double Linked List

Nama : Qanita Shafiyah

NIM : 1203230094

Kelas : IF-03-03

Deskripsi Tugas

Andi baru saja belajar tentang struktur data sirkular double linked list. Dia ingin melatih kemampuannya dengan membuat sebuah program untuk mengelola data dalam struktur tersebut. Andi ingin membuat sebuah program C yang akan meminta pengguna untuk memasukkan sejumlah data ke dalam sirkular double linked list. Setelah itu, program akan mengurutkan list tersebut secara ascending dengan syarat "Never change the data. Change the position of the nodes." Artinya, Anda dilarang mengubah/memanipulasi data pada setiap node, namun Anda dapat mengubah posisi dari node-node tersebut.

Penjelasan Program:

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

Header yang digunakan adalah ‘stdio.h’ dalam bahasa C mencakup input dan output. Fungsi header ‘stdlib.h’ yakni untuk alokasi memori dalam program ini menggunakan fungsi *‘malloc’* untuk mengalokasikan blok memori dan *‘free’* membebaskan blok memori atau menghapus blok memori yang dialokasikan sebelumnya.

```c
typedef struct Node {
    int data;
    struct Node *next, *prev;
} Node;
```

Tipe data *struct Node* dalam bahasa C digunakan untuk merepresentasikan node dalam sebuah linked list, yang dalam kasus ini adalah doubly linked list (linked list ganda) dimana terdapat beberapa variabel yaitu data yang bertipe integer, **struct Node next* yaitu pointer ke node berikutnya dalam linked list, dan **struct Node prev* yaitu pointer ke node sebelumnya dalam linked list. Ini memungkinkan traversal dua arah dalam linked list, baik maju (menggunakan next) maupun mundur (menggunakan prev).

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

Sebuah fungsi yang bertipe *node** yang bertugas untuk membuat node dengan parameter data yang bertipe integer. Dengan mendaklarasikan variabel newNode bertipe *node** merujuk pada struct sebelumnya dan mengalokasikan memori dengan menggunakan fungsi *'malloc'*. Kemudian nilai data dari newNode diisi dengan nilai parameter yang akan diberikan dan alamat memori yang ditunjuk dari node tersebut adalah dirinya sendiri karena ini merupakan *circular double linked list.*

```
16  void insertLast(Node** head, int data) {
17      if (*head == NULL) {
18          *head = createNode(data);
19          return;
20      }
21      Node *end = (*head)->prev;
22      Node* newNode = createNode(data);
23      newNode->next = *head;
24      (*head)->prev = newNode;
25      newNode->prev = end;
26      end->next = newNode;
27  }
```

Fungsi insertLast bertipe void ini bertugas untuk menambahkan node baru diakhir node dengan menggunakan parameter *node**head* dan data yang bertipe integer. Terdapat kondisi jika pointer head nilainya bernilai NULL maka otomatis akan memanggil fungsi createNode untuk membuat node. Jika tidak bernilai NULL maka program akan menginisiasi pembuatan node baru dan menambahkannya ke dalam linked list.

```
29  void printList(Node* head) {
30      if (head == NULL) {
31          printf("Linked list kosong\n");
32          return;
33      }
34      Node *curr = head;
35      do {
36          printf("Address: %p, Data: %d\n", (void*)curr, curr->data);
37          curr = curr->next;
38      } while (curr != head);
39  }
```

Fungsi printList bertipe void bertugas untuk menampilkan list yang telah ditambahkan oleh pengguna. Terdapat kondisi jika linked list kosong yang dimana head bernilai sama dengan NULL maka program akan mencetak bahwa Linked list kosong. Namun jika kondisi tersebut tidak terpenuhi maka akan dibuat variabel pembantu yaitu curr diinisiasikan dengan head kemudian dilakukan perulangan do-while yang dimana program akan mencetak alamat dari variabel tersebut beserta nilainya dan pernyataan do akan terus dijalankan selama nilai curr tidak sama dengan head.

```
41  void sortList(Node **head) {
42      if (*head == NULL) {
43          printf("Linked list kosong\n");
44          return;
45      }
46
47      int count = 0;
48      Node *temp = *head;
49      do {
50          count++;
51          temp = temp->next;
52      } while (temp != *head);
```

Fungsi sortList ini untuk mengurutkan linked list dengan node**head sebagai parameternya. Terdapat kondisi jika nilai pointer headnya bernilai NULL maka program akan mencetak Linked list kosong. Namun jika linked list memiliki sebuah nilai maka program menginisialisasi count ke 0 menggunakan temp untuk menjelajahi linked list mulai dari *head dan loop akan berjalan sampai kembali ke head, menghitung jumlah node.

```
54      Node **nodeArray = (Node **)malloc(count * sizeof(Node *));
55      temp = *head;
56      for (int i = 0; i < count; i++) {
57          nodeArray[i] = temp;
58          temp = temp->next;
59      }
60
```

Mengalokasikan memori untuk array pointer nodeArray yang berukuran count dan mengisi nodeArray dengan pointer ke setiap node dalam linked list.

```
61      for (int i = 0; i < count - 1; i++) {
62          for (int j = 0; j < count - i - 1; j++) {
63              if (nodeArray[j]->data > nodeArray[j + 1]->data) {
64                  Node *tempNode = nodeArray[j];
65                  nodeArray[j] = nodeArray[j + 1];
66                  nodeArray[j + 1] = tempNode;
67              }
68          }
69      }
70
```

Menggunakan algoritma bubble sort untuk mengurutkan nodeArray berdasarkan nilai data dari node.

```
71      for (int i = 0; i < count; i++) {
72          nodeArray[i]->next = nodeArray[(i + 1) % count];
73          nodeArray[(i + 1) % count]->prev = nodeArray[i];
74      }
75      nodeArray[0]->prev = nodeArray[count - 1];
76      nodeArray[count - 1]->next = nodeArray[0];
77
78      *head = nodeArray[0];
79      free(nodeArray);
80  }
```

Setiap node dalam nodeArray diatur untuk menunjuk ke node berikutnya (next) dan node sebelumnya (prev) dalam urutan yang telah diurutkan dan menggunakan operator modulus % untuk memastikan pointer berputar dalam linked list melingkar. *head diatur ke node pertama dalam nodeArray, yang sekarang adalah node dengan nilai terkecil setelah pengurutan dan membebaskan memori yang dialokasikan untuk nodeArray.

```
82  int main() {
83      int x, data;
84      Node *head = NULL;
85      printf("Masukkan jumlah data: ");
86      scanf("%d", &x);
87
88      for (int i = 0; i < x; i++) {
89          printf("Masukkan data ke-%d: ", i + 1);
90          scanf("%d", &data);
91          insertLast(&head, data);
92      }
93
```

Kemudian dalan fungsi main utama terdapat pendeklarasian variabel x dan data bertipe integer serta inisialisasi node kosong dengan *head diinisiasikan dengan NULL. Kemudian meminta inputan banyaknya data yang ingin dimasukkan oleh pengguna. Terdapat perulangan for dengan batas banyaknya jumlah inputan yang diinginkan pengguna dengan setiap perulangan akan meminta inputan nilai data yang nilainya akan dimasukkan dalam linked list dengan memanggil fungsi insertLast.

```
94      printf("\nList sebelum pengurutan:\n");
95      printList(head);
96      sortList(&head);
97      printf("\nList setelah pengurutan:\n");
98      printList(head);
99      return 0;
100 }
```

Kemudian program akan menampilkan list sebelum pengurutan dan memanggil fungsi printList kemudian menmanggil fungsi sortList dan ditampilkan lagi hasil linked list setelah pengurutan dengan memanggil fungsi printList.

Output :

```
Masukkan jumlah data: 5
Masukkan data ke-1: 7
Masukkan data ke-2: 8
Masukkan data ke-3: 5
Masukkan data ke-4: 4
Masukkan data ke-5: 2

List sebelum pengurutan:
Address: 00C429F0, Data: 7
Address: 00C42A08, Data: 8
Address: 00C42A20, Data: 5
Address: 00C42A38, Data: 4
Address: 00C42A50, Data: 2

List setelah pengurutan:
Address: 00C42A50, Data: 2
Address: 00C42A38, Data: 4
Address: 00C42A20, Data: 5
Address: 00C429F0, Data: 7
Address: 00C42A08, Data: 8
```