MODUL 5: DOUBLE LINKED LIST

1.1. Deskripsi Singkat

Pada praktikum ini, kita akan mencoba mempraktikkan pembuatan dan operasi-operasi pada Linear Double Linked List. Linear Double Linked List adalah Linked List lurus dengan pointer ganda, yaitu ada dua buah pointer. Jadi, dalam structure simpul, terdapat dua elemen yang bertipe pointer. Pointer pertama menunjuk atau berisi alamat simpul setelahnya (next node), dan pointer yang kedua menunjuk atau berisi alamat simpul sebelumnya (previous node).

1.2. Tujuan Praktikum

1) Mahasiswa mampu membuat Linear Double Linked List menggunakan bahasa C
2) Mahasiswa dapat melakukan operasi penyisipan maupun penghapusan simpul pada Linear Double Linked List dengan menggunakan bahasa C

1.3. Material Praktikum

Kegiatan pada modul ini memerlukan material berupa software editor dan compiler (atau IDE) untuk bahasa pemrograman C.

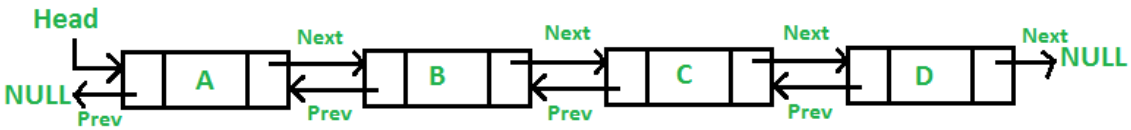
1.4. Kegiatan Praktikum

Jenis dan macam proses pada Linear Double Linked List sama saja dengan Linear Single Linked List yang telah kita coba pada praktikum sebelumnya, sehingga pengertian dari tiap-tiap proses dapat dibaca pada bahan praktikum sebelumnya.

A. Inisialisasi

1. Persiapan

Kita deklarasikan structure dengan nama **node**, yang terdiri dari **tiga** elemen. Elemen pertama yaitu value, yang berfungsi untuk menyimpan data dengan tipe integer. Elemen kedua adalah **prev**, yaitu pointer yang dipakai untuk menunjuk (mencatat alamat) structure **node** sebelumnya. Elemen ketiga adalah **next**, yaitu pointer yang dipakai untuk menunjuk (mencatat alamat) structure **node** setelahnya.



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

struct node{
    int value;
    struct node *next;
    struct node *prev;
};

typedef struct node *ptrnode;
```

Salin potongan program tersebut pada IDE Anda, lalu simpan dengan nama **praktikum5.c.** Tambahkan fungsi main pada potongan program tersebut, lalu jalankan.

2. Pembuatan simpul awal

Untuk menginisialisasi linked list, kita siapkan simpul paling kiri atau simpul awal atau head. Tambahkan potongan program berikut pada fungsi **`main()`**. Lalu jalankan ulang

```
ptrnode head = NULL;
ptrnode tail = NULL;
head = (ptrnode)malloc(sizeof(struct node));
tail = head;
head->value = 10;
head->next = NULL;
head->prev = NULL;
```

B. Pembuatan Sebuah Simpul

Cara pertama, deklarasikan **node** terlebih dahulu, baru kemudian deklarasikan **pointer**, dimana pointer diisi dengan alamat **node** yang telah sebelumnya dideklarasikan. Untuk mencoba cara ini, tambahkan potongan program berikut pada fungsi **`main()`**.

```
struct node node_dua;
ptrnode dua = &node_dua;
dua->value = 20;
dua->next = NULL;
dua->prev = NULL;
```

Agar simpul kedua yang baru saja kita buat tersambung (linked) dengan simpul head yang sebelumnya telah kita buat, maka tambahkan potongan program berikut, masih di dalam fungsi **`main().`**

```
head->next = dua;
dua->prev = head;
```

Cara pembuatan simpul yang kedua, deklarasikan pointer terlebih dahulu, baru kemudian deklarasikan **struct node**, sekaligus mengalokasikan memori untuk **struct node** tersebut dan menyimpannya ke dalam pointer yang telah dideklarasikan.

```
ptrnode tiga = NULL;
ptrnode empat = NULL;

tiga = (ptrnode)malloc(sizeof(struct node));
empat = (ptrnode)malloc(sizeof(struct node));

dua->next = tiga;

tiga->value = 30;
tiga->next = empat;
tiga->prev = dua;

empat->value = 40;
empat->next = NULL;
empat->prev = tiga;
```

Agar tidak perlu menuliskan potongan program di atas secara berulang-ulang setiap kali dibutuhkan untuk membuat node baru, maka potongan program untuk membuat simpul baru tersebut kita simpan di dalam sebuah fungsi, dalam hal ini kita beri nama **`createNode().`**

```
ptrnode createNode(int nilai){
ptrnode p;
p = (ptrnode)malloc(sizeof(struct node));
p->value = nilai;
p->next = NULL;
p->prev = NULL;
return(p);
}
```

Untuk membuat simpul baru, kita tinggal memanggil fungsi **createNode** tersebut di dalam fungsi **`main().`** Sebagai contoh, tambahkan potongan program berikut pada fungsi **`main()`**.

```
ptrnode lima = createNode(50);
```

```
empat->next = lima;
lima->prev = empat;
```

C. Menampilkan Nilai dari Linked List

- Apakah terdapat perbedaan algoritma untuk menampilkan nilai dari seluruh simpul yang terdapat pada sebuah Single Linked List dan Double Linked List?
- Bagaimana cara menampilkan nilai dari seluruh simpul yang terdapat pada sebuah Double Linked List dimulai dari node terakhir atau tail (*backward traversal*)?

D. Penambahan Simpul ke Dalam Linked List

1. Insert kiri, maksudnya menambahkan sebuah simpul baru diujung simpul paling kiri atau paling awal atau simpul head dari Linked List yang sudah ada. Tambahkan fungsi **`insert_head()`** berikut di luar fungsi **`main()`.**

```
ptrnode insert_head(ptrnode head, int nilai){
    ptrnode new_node = createNode(nilai);
    new_node->next = head;
    head->prev = new_node;
    head = new_node;

    return(head);
}
```

Kemudian coba panggil fungsi **`insert_head()`** dari fungsi **`main()`** dengan cara seperti di bawah ini.

```
/* Tambahkan node baru dengan value = 99 sebagai head */
    head = insert_head(head, 99);
```

2. Insert kanan, maksudnya menambahkan sebuah simpul baru diujung simpul paling kanan atau paling akhir atau simpul tail dari Linked List yang sudah ada. Tambahkan fungsi **`insert_tail()`** berikut di luar fungsi **`main()`**.

```
ptrnode insert_tail(ptrnode head, int nilai){
    /* iterasi mencari node terakhir*/
    ptrnode tail = head;
    while(tail->next != NULL)
        tail = tail->next;

    /* buat node baru */
```

```
    ptrnode new_node = createNode(nilai);
    tail->next = new_node;

    return(head);
}
```

Kemudian coba panggil fungsi **`insert_tail()`** dari fungsi **`main()`** dengan cara seperti di bawah ini.

```
/* Tambahkan node baru dengan value = 11 sebagai tail */
    head = insert_tail(head, 11);
```

PERHATIAN!!

Informasi simpul akhir atau tail akan sering dibutuhkan dalam penggunaan Double Linked List. Sebagaimana **pointer head**, Anda juga dapat mendeklarasikan **pointer tail** saat pembuatan simpul awal. Untuk memperbaharui posisi **pointer tail** tersebut, dapat dituliskan potongan program tambahan pada fungsi **insert kanan** dan **delete kanan**. Dengan demikian, **pointer tail** dapat langsung digunakan setiap kali dibutuhkan dan program tidak perlu melakukan iterasi untuk mencari simpul akhir atau tail.

3. Menambahkan simpul baru setelah simpul tertentu. Tambahkan potongan program berikut di luar fungsi **`main()`**.

```
ptrnode insert_after(ptrnode head, int nilai, int
nilai_dicari){
     /* cari node sebelumnya, starting from the first node*/
    ptrnode cursor = head;
    while(cursor->value != nilai_dicari)
        cursor = cursor->next;

    ptrnode new_node = createNode(nilai);
    new_node->next = cursor->next;
    cursor->next->prev = new_node;
    new_node->prev = cursor;
    cursor->next = new_node;

    return(head);
}
```

Kemudian coba panggil fungsi tersebut dari fungsi **`main()`** dengan cara seperti di bawah ini, lalu coba jalankan.

```
/* Tambahkan node baru dengan value = 60 setelah node dengan
value 50 */
head = insert_after(head, 60, 50);
```

4. Menambahkan simpul baru sebelum simpul tertentu. Tambahkan potongan program berikut di luar fungsi **`main()`**.

```
ptrnode insert_before(ptrnode head, int nilai, int
next_nilai){
   if (head->value == next_nilai)
       head = insert_head(head, nilai);
   else
   {

//pencarian nilai sama seperti insert after, tidak perlu 2
cursor
        ptrnode cursor = head;
        while(cursor->value != nilai_dicari)
        cursor = cursor->next;

        ptrnode new_node = createNode(nilai);
        new_node->prev = cursor->prev;
        cursor->prev->next = new_node;
        new_node->next = cursor;
        cursor->prev = new_node;
   }
   return(head);
}
```

Kemudian coba panggil fungsi tersebut dari fungsi **`main()`** dengan cara seperti di bawah ini, lalu coba jalankan.

```
/* Tambahkan node baru dengan value = 35 sebelum node dengan
value 40 */
head = insert_before(head, 35, 40);
```

Untuk melihat kembali isi keseluruhan simpul setelah beberapa penambahan simpul yang telah kita lakukan**,** pada fungsi **`main()`** panggil kembali fungsi untuk menampilkan nilai dari Linked List yang telah Anda buat sebelumnya pada Kegiatan Praktikum C.

E. Penghapusan Simpul dari Linked List

1. Delete kiri atau delete awal, maksudnya adalah menghapus simpul yang ada di ujung paling kiri atau paling awal atau simpul head dari Linked List yang sudah ada. Tambahkan fungsi **`remove_first()`** berikut di luar fungsi **`main()`.**

```
ptrnode remove_first(ptrnode head){
    if(head == NULL)
        return;

    ptrnode first = head;
    head = head->next;
    head->prev = NULL;
    first->next = NULL;

    free(first);

    return(head);
}
```

Kemudian coba panggil fungsi **`remove_first()`** dari fungsi **`main()`** dengan cara seperti di bawah ini.

```
/* Hapus node head */
head = remove_first(head);
```

2. Delete kanan atau delete akhir, maksudnya adalah menghapus simpul yang ada di ujung paling kanan atau paling akhir atau simpul tail dari Linked List yang sudah ada. Tambahkan fungsi **`remove_last()`** berikut di luar fungsi **`main()`.**

```
ptrnode remove_last(ptrnode head){
    if(head == NULL)
        return;

    //cursor bantuan satu lagi (prev_tail) tidak dibutuhkan
    ptrnode tail = head;
    while(tail->next != NULL)
    {
        tail = tail->next;
    }
```

```
    tail->prev = NULL;
    tail->prev->next = NULL;
    free(tail);
    return(head);
}
```

Kemudian coba panggil fungsi **remove_last()** dari fungsi **main()** dengan cara seperti di bawah ini.

```
/* Hapus node tail */
head = remove_last(head);
```

3. Delete tengah, maksudnya yaitu menghapus sebuah simpul yang berada di antara dua buah simpul lain. Bukan menghapus simpul yang paling kiri dan juga bukan menghapus simpul yang paling kanan. Tambahkan potongan program berikut di luar fungsi **main()**.

```
ptrnode remove_middle(ptrnode head, int nilai){
    ptrnode cursor = head;
    while(cursor != NULL)
    {
        if(cursor->next->value == nilai)
            break; //keluar dari iterasi
        cursor = cursor->next;
    }

    if(cursor != NULL)
    {
        ptrnode tmp = cursor->next;
        cursor->next = tmp->next;
        tmp->next->prev = cursor;
        tmp->next = NULL;
        tmp->prev = NULL;
        free(tmp);
    }

    return(head);
}
```

Kemudian coba panggil fungsi tersebut dari fungsi **`main()`** dengan cara seperti di bawah ini, lalu coba jalankan.

```
/* Hapus node di tengah */
head = remove_middle(head, 30);
```

Menggunakan salah satu cara penghapusan simpul di atas, kita dapat menghapus keseluruhan simpul yang ada pada Linked List.

Setelah fungsi-fungsi hapus di atas dijalankan, panggil kembali fungsi untuk menampilkan nilai linked list pada fungsi **`main()`,** untuk melihat apa isi yang tersisa pada Linked List.

1.5. Penugasan

1. Cobalah untuk memodifikasi potongan program pada pembuatan simpul awal, insert kanan, dan delete kanan sehingga pointer tail dideklarasikan dan selalu diperbaharui isinya saat penambahan dan penghapusan simpul dari kanan.

2. Buat sebuah program untuk menampilkan output di bawah ini menggunakan double linked list!

```
Input the number of nodes : 3
Input data for node 1 : 2
Input data for node 2 : 5
Input data for node 3 : 8

Data entered in the list are :
node 1 : 2
node 2 : 5
node 3 : 8
Input data for the first node : 1

After insertion the new list are :
node 1 : 1
node 2 : 2
node 3 : 5
node 4 : 8
```

3. Bagaimana untuk membalik nilai-nilai dalam double linked list (tail ke head)?