MODUL 4: SINGLE LINKED LIST

1.1. Deskripsi Singkat

Struktur data, dilihat dari alokasi memori yang digunakan, dibagi menjadi 2 yaitu dinamis dan statis. Pada struktur statis, elemen data bersifat tetap/fixed, artinya ukuran dan urutannya sudah pasti. Contoh struktur statis adalah array. Alokasi memori pada array akan terbuang jika array tidak diisi penuh, dan bermasalah ketika harus menambah ukuran yang ditetapkan di awal. Selain itu, ruang memori yang dipakai oleh array tidak dapat dihapus bila array tersebut sudah tidak digunakan lagi pada saat program dijalankan. Untuk memecahkan masalah di atas, kita dapat menggunakan linked list. Linked List merupakan jenis struktur data yang dinamis. Struktur dinamis memungkinkan jumlah elemen data dapat berubah-ubah (tambah atau kurang) pada saat program dijalankan.

Linked List adalah struktur berupa rangkaian objek saling berkait (linked). Masing-masing objek sering disebut dengan **simpul** atau **node** atau **verteks**. Objek itu sendiri adalah merupakan gabungan beberapa elemen data (variabel) yang dijadikan satu kelompok dalam bentuk **structure**. Untuk menghubungkan objek satu dengan objek lainnya, diperlukan paling tidak sebuah variabel **pointer**. Penggunaan pointer berakibat objek-objek bersebelahan secara logic walaupun tidak bersebelahan secara fisik di memori. Terdapat beberapa jenis linked list, yaitu: **Linear Single Linked List**, **Linear Double Linked List**, **Circular Single Linked List**, dan **Circular Double Linked List**. Pada praktikum ini, kita akan mencoba mempraktikkan pembuatan dan operasi-operasi pada Linear Single Linked List.

Linear Single Linked List merupakan Linked List yang paling sederhana. Setiap simpul dibagi menjadi dua bagian yaitu bagian isi dan bagian pointer. Bagian isi merupakan bagian yang berisi data yang disimpan oleh simpul, sedangkan bagian pointer merupakan bagian yang berisi alamat dari simpul berikutnya.

1.2. Tujuan Praktikum

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

Pada penggunaan Linked List, ada 4 macam proses dasar, yaitu:

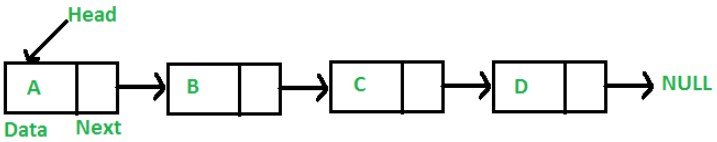
1. Inisialisasi
2. Membuat simpul awal
3. Membuat simpul baru dan menambahkannya ke dalam Linked List

4. Menghapus simpul dari Linked List

A. Inisialisasi

1. Persiapan

Untuk pembuatan linked list, kita memerlukan suatu structure yang nantinya akan digunakan untuk pembuatan simpul. Dalam potongan program di bawah ini, kita deklarasikan structure dengan nama **node**, yang terdiri dari dua elemen. Elemen pertama yaitu value, yang berfungsi untuk menyimpan data dengan tipe integer. Elemen kedua adalah **next**, yang bertipe pointer yang dipakai untuk menunjuk (mencatat alamat) structure **node** lainnya. Selain pointer **next** yang merupakan bagian dari structure **node**, kita memerlukan pointer lainnya untuk menunjuk simpul pada linked list yang akan kita buat. Perhatikan ilustrasi Linear Single Linked List berikut.



Dari ilustrasi di atas, terdapat pointer **Head** yang menunjuk simpul paling kiri atau simpul awal. Selain pointer **Head**, kita akan memerlukan pointer-pointer lain untuk membantu dalam mengoperasikan Linked List. Oleh karena itu, kita juga mendeklarasikan sebuah pointer dengan tipe data **struct node**, dalam hal ini dimisalkan dengan nama **ptrnode.**

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

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

typedef struct node *ptrnode;
```

Salin potongan program tersebut pada IDE Anda, lalu simpan dengan nama **praktikum4.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;
```

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

B. Pembuatan Sebuah Simpul

Linked list adalah kumpulan dari simpul-simpul. Simpul-simpul tersebut dibuat satu per satu, tidak sekaligus. Selain simpul awal, maka setiap simpul yang baru dibuat, harus dihubungkan (linked) dengan salah satu simpul yang sudah ada. Terdapa beberapa cara untuk membuat simpul baru. Cara pertama, kita mendeklarasikan **node** terlebih dahulu. Baru kemudian kita mendeklarasikan **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;
```

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;
```

Cara pembuatan simpul yang kedua, sama seperti pembuatan simpul **head** pada tahapan inisialisasi. Kita deklarasikan pointer terlebih dahulu, baru kemudian kita deklarasikan **struct node**, sekaligus mengalokasikan memori untuk **struct node** tersebut dan menyimpannya ke dalam pointer yang telah dideklarasikan. Tambahkan potongan program berikut pada fungsi **`main()`**. Pada potongan program ini, kita akan membuat dua simpul baru, dan menghubungkannya dengan kedua simpul yang telah dibuat sebelumnya.

```
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;
```

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

Cara ketiga, 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. Tambahkan potongan program berikut di luar fungsi **`main()`**, tepatnya setelah deklarasi **struct node** dan **pointer *ptrnode.**

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

C. Menampilkan Nilai dari Linked List

Sebelum kita lanjutkan ke materi praktikum berikutnya, cobalah untuk menampilkan nilai dari simpul-simpul yang telah kita buat pada Kegiatan Praktikum A dan B. Algoritma untuk menampilkan nilai dari seluruh simpul yang terdapat pada sebuah Linked List adalah sebagai berikut:

1. Iterasi setiap node dalam sebuah linked list. Gunakan pointer bantuan untuk memeriksa apakah terdapat **node** ataukah **NULL**.
2. Lakukan **`printf()`** elemen data dari setiap node mulai dari head, node berikutnya, dan seterusnya sampai node tersebut NULL.

Apakah Anda berhasil menampilkan seluruh nilai pada simpul pertama hingga simpul kelima?

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 = 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);
```

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

```
ptrnode insert_after(ptrnode head, int nilai, int prev_nilai){
```

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

    ptrnode new_node = createNode(nilai);
    new_node->next = cursor->next;
    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
   {
       ptrnode cursor, prevcursor;
       cursor = head;
       do
       { prevcursor = cursor;
         cursor = cursor->next;
       }
       while (cursor->value != next_nilai);

       ptrnode new_node = createNode(nilai);
       new_node->next = cursor;
       prevcursor->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 = 35 sebelum node dengan
value 99 */
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;
    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;

    ptrnode tail = head;
    ptrnode prevtail = NULL;
    while(tail->next != NULL)
    {
        prevtail = tail;
        tail = tail->next;
    }

    prevtail->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 = 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);
```

4. Menggunakan salah satu cara penghapusan simpul di atas, kita dapat menghapus keseluruhan simpul yang ada pada Linked List. Sebagai contoh, program di bawah ini menjalankan algoritma yang sama dengan fungsi **`remove_first()`**, hanya ditambahkan iterasi agar proses penghapusan dilakukan secara berulang-ulang untuk keseluruhan simpul yang ada pada Linked List. Tambahkan fungsi **`dispose()`** berikut di luar fungsi **`main()`.**

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

    while(head != NULL){
        ptrnode tmp = head;
        head = head->next;

        tmp->next = NULL;
        free(tmp);
    }

    printf("semua node telah dihapus\n");
    return(head);
}
```

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

```
/* Hapus/free linked list */
head = dispose(&head);
```

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. Buatlah fungsi untuk menghitung jumlah node dalam sebuah linked list! (looping sama seperti pada saat menampilkan nilai dari linked list).

2. Buatlah fungsi untuk membalik nilai dari head ke tail!
   Contoh: 5->4->3->2->1 menjadi 1->2->3->4->5
   Hint:
   - hanya nilai saja, memory address (pointer node) tetap sama
   - buat temporary pointer node sebagai bantuan: prev, current, next dan loop dari head ke tail
3. Buat program untuk menyimpan data students berisi **int nim, char nama[50]** secara dinamis!