Praktikum 5

Single Linked List (1)

A. TUJUAN PEMBELAJARAN

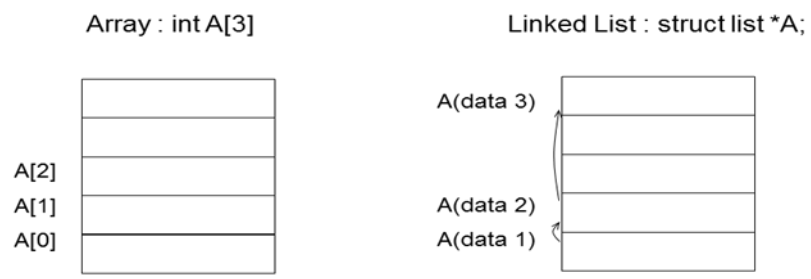
Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

1. Memahami konsep membangun *single linked list*
2. Memahami konsep operasi menyisipkan sebagai simpul ujung(awal) dari *linked list*
3. Memahami konsep operasi membaca atau menampilkan
4. Memahami konsep operasi mencari sebuah simpul tertentu
5. Memahami konsep operasi menyisipkan sebagai simpul terakhir
6. Mengimplementasikan semua operasi *single linked list* dalam pemrograman

B. DASAR TEORI

Secara konseptual Array adalah struktur data yang memesan tempat secara berurutan untuk jumlah data statis. Sedangkan *Memory Allocation* (*malloc*) adalah sebuah fungsi fasilitas untuk memesan tempat secara berurutan untuk tipe data pointer dengan jumlah data dinamis.

Permasalahan akan timbul jika kita memesan data dengan jumlah yang besar sementara tempat di memori yang berurutan tidak ada. Solusi untuk permasalahan ini adalah dengan menggunakan *linked list* atau senara berantai. *Linked list* merupakan deretan elemen yang berdampingan. Akan tetapi, karena elemen-elemen tersebut dialokasikan secara dinamis (menggunakan *malloc*), sangat penting untuk diingat bahwa kenyataannya, *linked list* akan terpencar-pencar di memori. Pointer pada suatu elemen berisi alamat untuk data berikutnya sebagai penjamin bahwa semua elemen dapat diakses. Pada Gambar 5.1 ditunjukkan perbedaan array dan *Linked list*.



Gambar 5.1 Perbedaan Penempatan di Memory untuk Array dan *Linked List*

B.1 Bagaimana Membangun *Linked List*

Untuk membangun sebuah *linked list*, maka terdapat beberapa langkah yang harus disiapkan. Langkah tersebut adalah sebagai berikut

1. Deklarasi
2. Alokasi memori
3. Mengisi data
4. Menyiapkan untuk dihubungkan dengan data baru berikutnya

B.1.1 Deklarasi Simpul

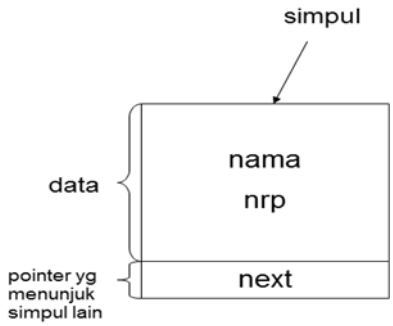
Linked list terdiri dari elemen-elemen individu, dimana masing-masing dihubungkan dengan pointer tunggal. Masing-masing elemen terdiri dari dua bagian, yaitu sebuah data dan sebuah pointer yang disebut dengan pointer *next*. Dengan menggunakan struktur *two-member* seperti ini, *linked list* dibentuk dengan cara menunjuk pointer *next* suatu elemen ke elemen yang mengikutinya. Pointer *next* pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list.

Pada Gambar 5.2 ditunjukkan deklarasi untuk sebuah simpul menggunakan struktur yang diikuti deklarasi sebuah variable yang menggunakan struktur tersebut.

```
struct simpul
{
      char nama[25];
      int nrp;
      struct simpul *next;
};
struct simpul *ujung;
```

Gambar 5.2 Deklarasi Simpul dalam *Linked list*

Dari Gambar 5.2 didapatkan gambaran sebuah simpul pada Gambar 5.3



Gambar 5.3 Gambaran Simpul yang Dibangun dengan Deklarasi pada Gambar 5.2

B.1.2 Alokasi Memori

Ketika sebuah variabel dideklarasikan, terlebih dahulu harus diinisialisasi. Demikian juga dengan pengalokasian secara dinamis. Sehingga, fungsi untuk mengalokasikan sebuah node baru, fungsi `alokasi_simpul()` menggunakan `malloc()` untuk mendapatkan memori aktual, yang akan menginisialisasi suatu field data. next selalu diinisialisasi sebagai NULL.

Untuk melihat kemungkinan alokasi memori gagal, maka fungsi `alokasi_simpul()` menghasilkan NULL, bila berhasil maka menghasilkan sebuah simpul. Fungsi dari `alokasi_simpul()` adalah sebagai berikut :

```
struct simpul* alokasi_simpul()
{
      struct simpul * new;
      new = (struct simpul*)malloc(sizeof(struct simpul));
      if(new==NULL)
            return NULL;
      else
            new->next=NULL;
      return new;
}
```

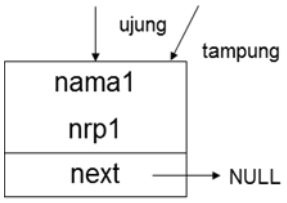
B.1.3 Mengisi Data Dan Menyiapkan Dihubungkan Dengan Data Berikutnya

Setelah melakukan deklarasi untuk simpul dan terdapat fungsi untuk melakukan alokasi memori maka keduanya kita gunakan untuk membangun sebuah *linked list* dengan prinsip LIFO (*Last In First Out*) dengan perintah sebagai berikut:

```
struct simpul *ujung;
ujung=alokasi_simpul();
if(ujung==NULL)printf("Alokasi gagal");
else
{
      printf("Nama      :");scanf("%s",&ujung->nama);
      printf("NRP :");scanf("%d",&ujung->nrp);
      if(j==0)/*kondisi jika data masih kosong*/
           tampung=ujung;
}
```

Gambar 5.4 Perintah untuk Membangun *Linked list*

Dengan perintah pada Gambar 5.4 didapatkan ilustrasi alokasi memori untuk sebuah simpul seperti Gambar 5.5.



Gambar 5.5 Simpul yang Terbentuk dengan Perintah pada Gambar 5.4

B.2 Operasi Pada *Linked list*

Terdapat beberapa Operasi yang penting pada *linked list*, yaitu:

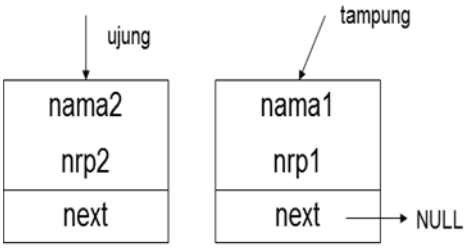
1. Menyisipkan sebagai simpul ujung(awal) dari *linked list*.
2. Membaca atau menampilkan
3. Mencari sebuah simpul tertentu
4. Menyisipkan sebagai simpul terakhir
5. Menghapus simpul tertentu
6. Menyisipkan setelah simpul tertentu
7. Menyisipkan sebelum simpul tertentu

B.2.1 Menyisipkan Sebagai Simpul Ujung(Awal) Dari *Linked list*

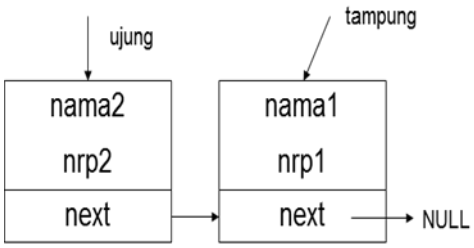
Setelah mendeklarasikan, mengisi data dan menyiapkan untuk dihubungkan dengan simpul berikutnya untuk menciptakan simpul pertama seperti yang dilakukan sebelumnya. Kita bisa melanjutkan dengan operasi menyisipkan sebagai simpul ujung dari *linked list*. Pada Gambar 5.6 merupakan perintah untuk menyisipkan simpul kedua dst sebagai simpul paling ujung dari *linked list.*

```
1. ujung=alokasi_simpul();
2. if(ujung==NULL)printf("Alokasi gagal");
3. else
4. {
5. printf("Nama      :");scanf("%s",&ujung->nama);
6. printf("NRP :");scanf("%d",&ujung->nrp);
7. if(j==0)/*kondisi jika data masih kosong*/
8. tampung=ujung;
9. else
10. {
11. ujung->next=tampung;
12. tampung=ujung;
13. }
14. }
```

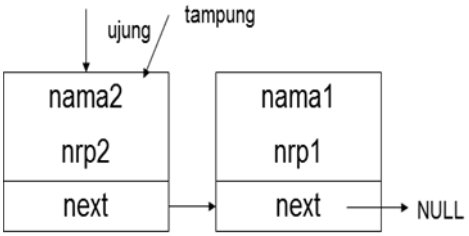
Setelah perintah baris ke-5



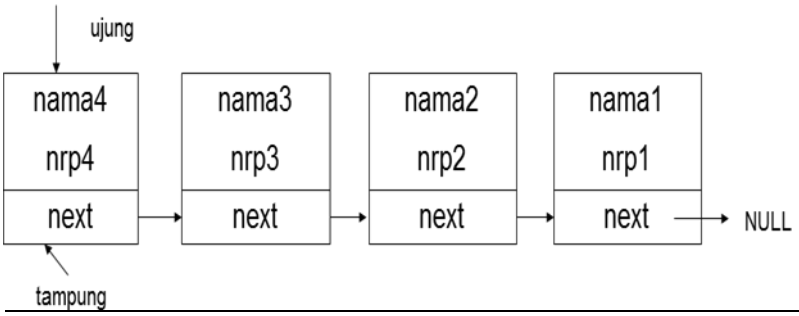
Setelah perintah baris ke-6-10



Setelah perintah baris ke-11



Setelah iterasi ke-4



B.2.2 Membaca Atau Menampilkan

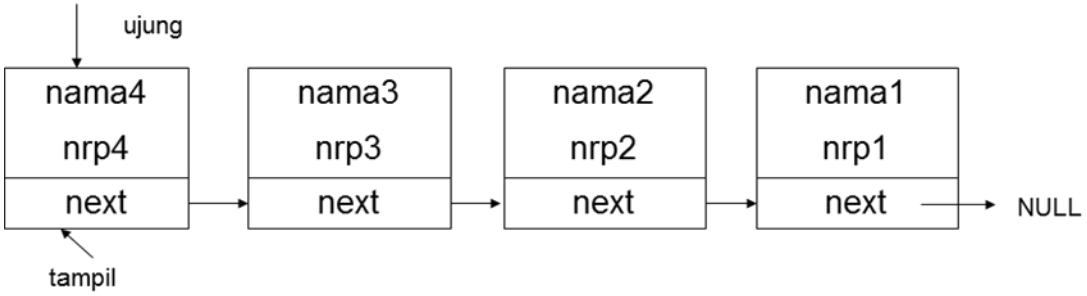
Langkah-langkah untuk membaca atau menampilkan *linked list* yang sudah terbentuk di atas adalah sebagai berikut:

1. Inisialisasi sebuah variabel bertipe `struct simpul* (tampil)` dengan ujung
2. Tampilkan data nama dan nrp yang ada pada tampil
3. Jalankan `tampil = tampil->next`
4. Ulangi langkah 2 selama tampil tidak sama dengan NULL

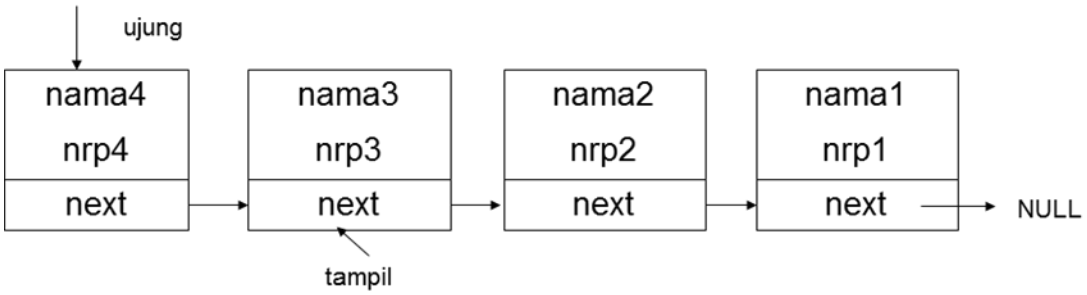
Berikut ini adalah perintah untuk membaca atau menampilkan data pada *single linked list*

```
1. tampil = ujung;
2. while (tampil!=NULL)
3. {
4.      printf("%s",tampil->nama);
5.      printf("%d",tampil->nrp);
6.      tampil = tampil -> next;
7. }
```

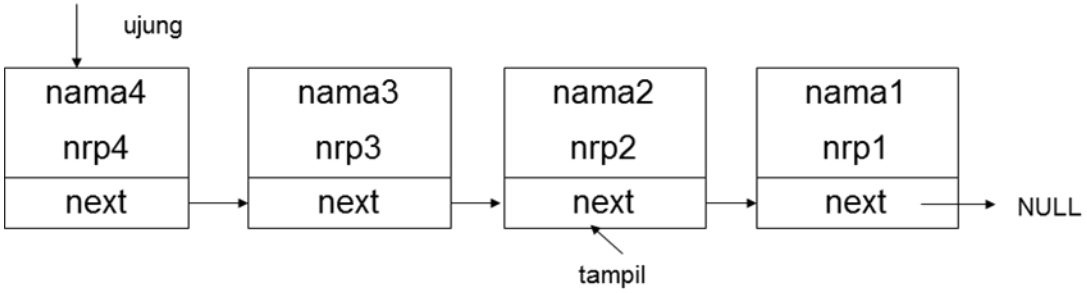
Setelah perintah baris 1



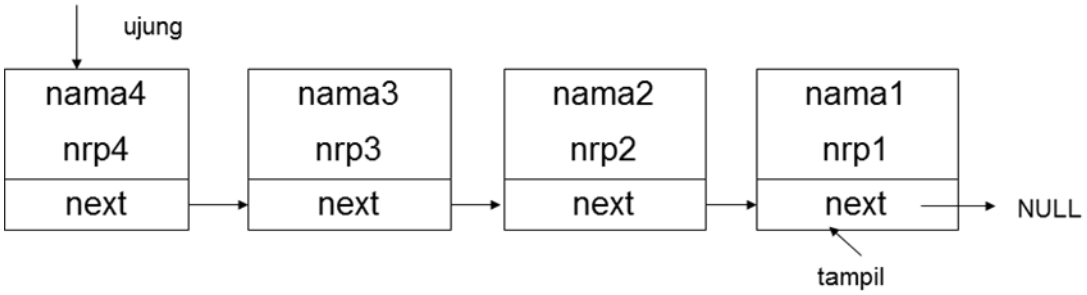
Setelah perintah baris ke-6 iterasi pertama



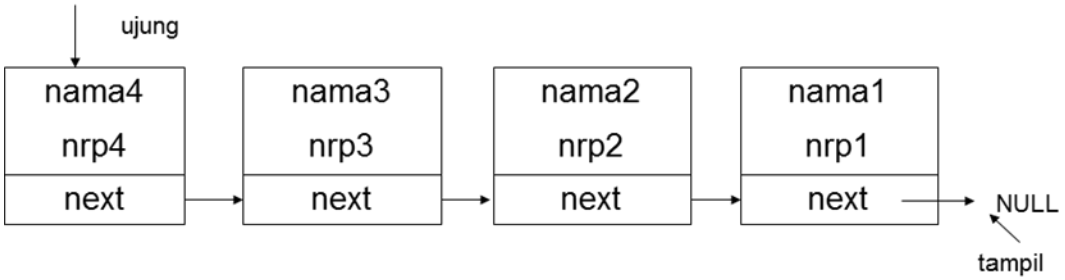
Setelah perintah baris ke-6 iterasi kedua



Setelah perintah baris ke-6 iterasi ketiga



Setelah perintah baris ke-6 iterasi keempat dan selesai



B.2.3 Mencari Simpul Tertentu

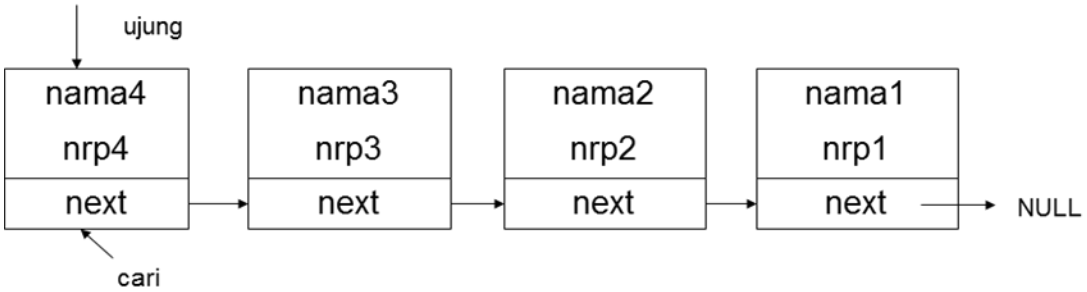
Langkah-langkah untuk mencari simpul tertentu pada *linked list* yang sudah terbentuk di atas adalah sebagai berikut:

1. Inisialisasi sebuah variabel bertipe `struct simpul* (cari)` dengan `ujung`
2. Ulangi langkah 3 jika data pada simpul `cari` tidak sama dengan data yang dicari atau cari tidak sama NULL
3. Jalankan `cari = cari->next`

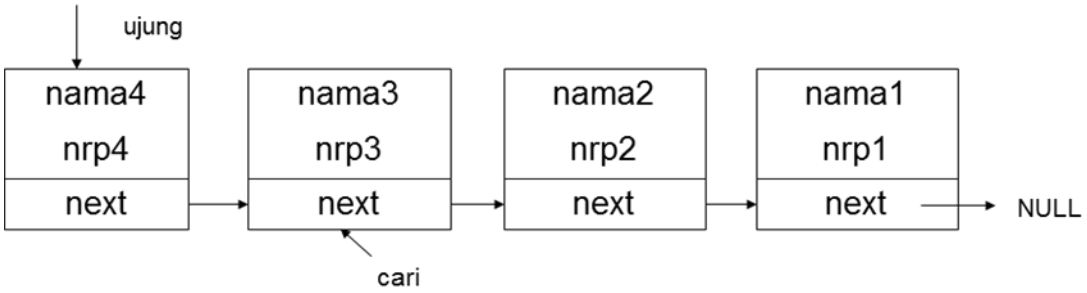
Berikut ini adalah perintah untuk mencari sebuah data pada *single linked list*

```
1.  cari = ujung;
2.  while (cari->nama!=nama2)
3.      cari = cari->next;
```

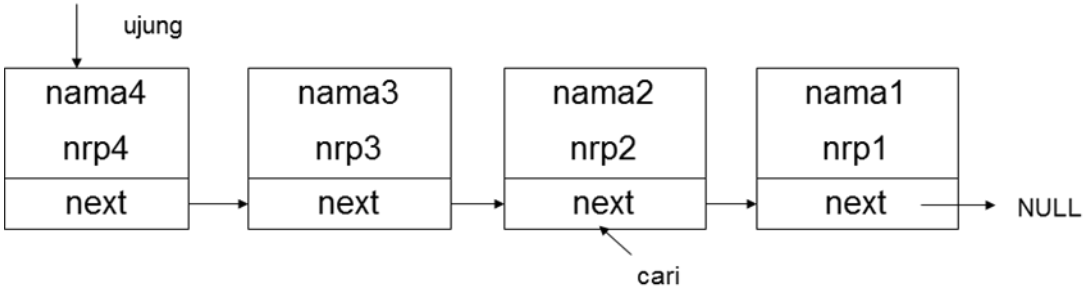
Setelah perintah baris 1



Setelah iterasi ke-2



Setelah iterasi sampai pada data yang dicari



B.2.4 Menyisipkan Sebagai Simpul Terakhir

Langkah-langkah untuk menyisipkan simpul baru sebagai simpul terakhir pada *linked list* yang sudah terbentuk di atas adalah sebagai berikut:

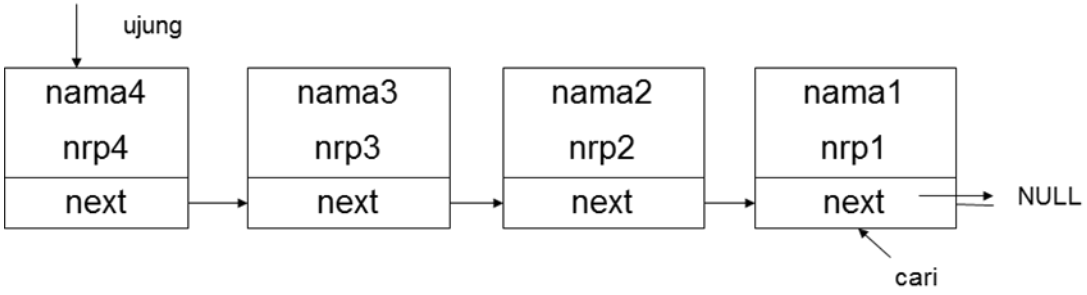
1. Alokasikan memori untuk simpul baru yang akan disisipkan
2. Inisialisasi sebuah variabel bertipe `struct simpul* (cari)` dengan `ujung`
3. Lakukan proses pengulangan pencarian sampai `cari->next` sama dengan NULL
4. Hubungkan `cari->next` ke simpul `baru`

Berikut ini adalah perintah untuk menyisipkan data baru sebagai simpul terakhir pada *single linked list*

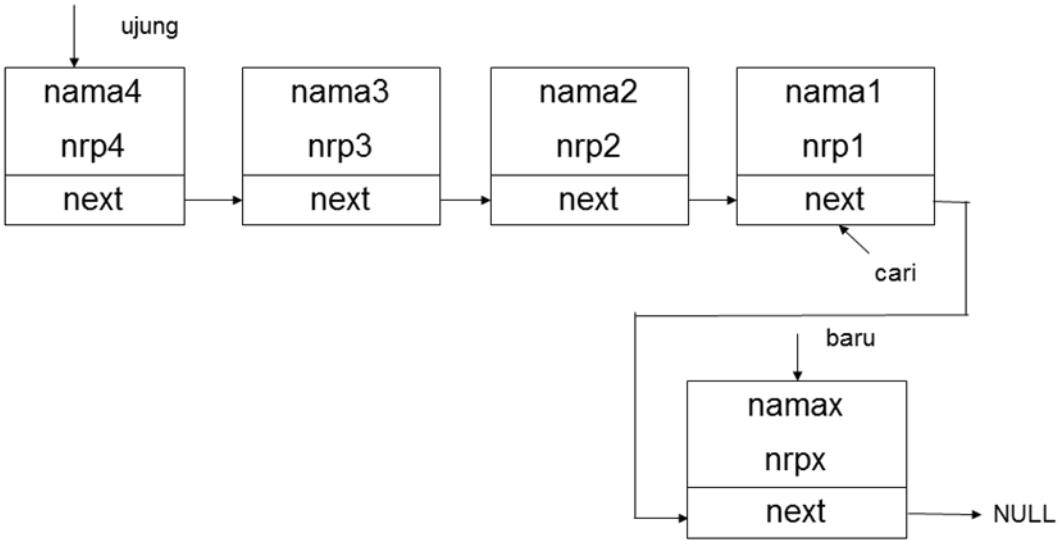
```
1. baru=alokasi_simpul();
```

```
2. cari = ujung;
3. while (cari->next !=NULL)
4.      cari = cari->next;
5. cari->next=baru;
```

Setelah proses pencarian di mana cari->next = NULL



Setelah perintah baris ke-4



C. TUGAS PENDAHULUAN

Untuk semua operasi dasar *single linked list* persoalan di bawah ini, desainlah algoritma dan flowchartnya :

1. Menyisipkan sebagai simpul ujung(awal) dari *linked list*.
2. Membaca atau menampilkan
3. Mencari sebuah simpul tertentu
4. Menyisipkan sebagai simpul terakhir

D. PERCOBAAN

1. Implementasikan operasi dasar *Single linked list* : Menyisipkan sebagai simpul ujung(awal) dari *linked list*.
2. Implementasikan operasi dasar *Single linked list* : Membaca atau menampilkan
3. Implementasikan operasi dasar *Single linked list* : Mencari sebuah simpul tertentu. Tambahkan kondisi jika yang dicari adalah data yang paling depan.
4. Implementasikan operasi dasar *Single linked list* : Menyisipkan sebagai simpul terakhir
5. Gabungkan semua operasi di atas dalam sebuah Menu Pilihan.

E. LATIHAN

Bangunlah *Single linked list* di atas adalah *Single linked list* dengan prinsip LIFO.

F. LAPORAN RESMI

1. Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.
2. Tuliskan kesimpulan dari percobaan dan latihan yang telah anda lakukan.