MODUL 6
SINGLE & DOUBLE LINKED LIST

1. Tujuan Instruksional Umum

a. Mahasiswa dapat melakukan perancangan aplikasi menggunakan struktur Linked List (Senarai Berkait)

b. Mahasiswa mampu melakukan analisis pada algoritma *Single & Double Linked List* yang dibuat

c. Mahasiswa mampu mengimplementasikan algoritma *Single & Double Linked List* pada sebuah aplikasi secara tepat dan efisien

2. Tujuan Instruksional Khusus

a. Mahasiswa dapat menjelaskan mengenai *Single & Double Linked List*

b. Mahasiswa dapat membuat dan mendeklarasikan Abstraksi Tipe Data *Single & Double Linked List*

c. Mahasiswa mampu menerapkan operasi *Single Linked List Non Circular & Single Linked List Circular*

d. Mahasiswa mampu menerapkan *Double Linked List Non Circular & Double Linked List Circular*

Pengertian Linked List

Salah satu bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis dan terbatas adalah senarai berkait (*linked list*). Suatu senarai berkait (*linked list*) adalah suatu simpul (*node*) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. Suatu simpul dapat berbentuk suatu struktur atau *class*. Simpul harus mempunyai satu atau lebih elemen struktur atau *class* yang berisi data. Secara teori, *linked list* adalah sejumlah node yang dihubungkan secara linier dengan bantuan *pointer*. Senarai berkait lebih efisien di dalam melaksanakan penyisipan-penyisipan dan penghapusan-penghapusan. Senarai berkait juga menggunakan alokasi penyimpanan secara dinamis, yang merupakan penyimpanan yang dialokasikan pada *runtime*. Karena di dalam banyak aplikasi, ukuran dari data itu tidak diketahui pada saat kompile, hal ini bisa merupakan suatu atribut yang baik juga. Setiap *node* akan berbentuk *struct* dan memiliki satu buah *field* bertipe *struct* yang sama, yang berfungsi sebagai *pointer*. Dalam menghubungkan setiap node, kita dapat menggunakan cara *first-create-first-access* ataupun *first-create-last-access*. Yang berbeda dengan deklarasi *struct* sebelumnya adalah satu *field* bernama *next*, yang bertipe *struct tnode*. Hal ini sekilas dapat membingungkan. Namun, satu hal yang jelas, variabel *next* ini akan menghubungkan kita dengan *node* di sebelah kita, yang juga bertipe *struct tnode*. Hal inilah yang menyebabkan *next* harus bertipe *struct tnode*.

Bentuk Umum :

```
typedef struct telmtlist
    {
      infotype info;
      address next;
    }elmtlist;
```

infotype → sebuah tipe terdefinisi yang menyimpan informasi sebuah elemen list
next → address dari elemen berikutnya (suksesor) .
Jika L adalah list, dan P adalah address, maka alamat elemen pertama list L dapat diacu dengan notasi : `first (L)`
Sebelum digunakan harus dideklarasikan terlebih dahulu :

```
#define First(L) (L)
```

Elemen yang diacu oleh P dapat dikonsultasi informasinya dengan notasi :

`info(P)` deklarasi `#define info(P) P->info`

`next(P)` deklarasi `#define next(P) P->next`

Beberapa Definisi :
1. List l adalah list kosong, jika First(L) = Nil
2. Elemen terakhir dikenali, dengan salah satu cara adalah karena Next(Last) = Nil
Nil adalah pengganti Null, perubahan ini dituliskan dengan #define Nil Null

Operasi-operasi Linked List

- ***Insert***
  Istilah Insert berarti menambahkan sebuah simpul baru ke dalam suatu linked list.
- ***IsEmpty***
  Fungsi ini menentukan apakah linked list kosong atau tidak.
- ***Find First***
  Fungsi ini mencari elemen pertama dari linked list.
- ***Find Next***
  Fungsi ini mencari elemen sesudah elemen yang ditunjuk now.
- ***Retrieve***
  Fungsi ini mengambil elemen yang ditunjuk oleh now. Elemen tersebut lalu dikembalikan oleh fungsi.
- ***Update***
  Fungsi ini mengubah elemen yang ditunjuk oleh now dengan isi dari sesuatu.
- ***Delete Now***
  Fungsi ini menghapus elemen yang ditunjuk oleh now. Jika yang dihapus adalah elemen pertama dari linked list (head), head akan berpindah ke elemen berikutnya.
- ***Delete Head***
  Fungsi ini menghapus elemen yang ditunjuk head. Head berpindah ke elemen sesudahnya.
- ***Clear***
  Fungsi ini menghapus linked list yang sudah ada. Fungsi ini wajib dilakukan bila anda ingin mengakhiri program yang menggunakan linked list. Jika anda melakukannya, data-data yang dialokasikan ke memori pada program sebelumnya akan tetap tertinggal di dalam memori.

a. Single Linked List (Senarai berkait tunggal)

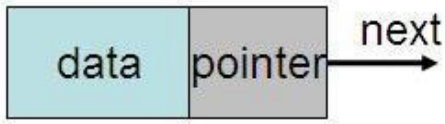
Single linked list adalah apabila hanya ada satu pointer yang menghubungkan setiap node (satu arah “next”).

a.1. Single Linked List non Circular

Pembuatan struct bernama tnode berisi 2 field, yaitu field data bertipe integer dan

field next yang bertipe pointer dari tnode.
Deklarasi node dengan struct:

```
struct tnode
{
int data;
struct tnode *next;
```

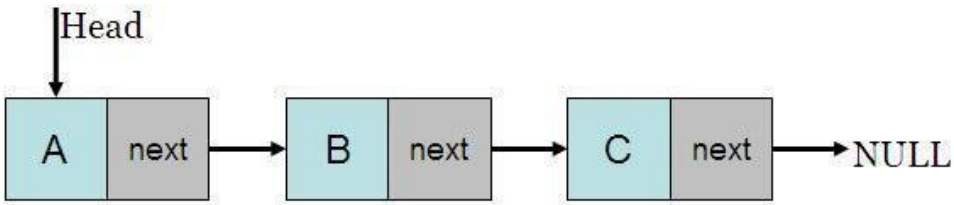


Gambar 1. Sebuah node pada Single Linked List

Asumsikan kita memiliki sejumlah *node* yang selalu menoleh ke sebelah dalam arah yang sama. Atau, sebagai alat bantu, kita bisa mengasumsikan beberapa orang yang bermain kereta api. Yang belakang akan memegang yang depan, dan semuanya menghadap arah yang sama. Setiap pemain adalah *node*. Dan tangan pemain yang digunakan untuk memegang bahu pemain depan adalah variabel *next*. Sampai di sini, kita baru saja mendeklarasikan tipe data dasar sebuah *node*. Selanjutnya, kita akan mendeklarasikan beberapa variabel *pointer* bertipe *struct tnode*. Beberapa variabel tersebut akan kita gunakan sebagai awal dari *linked list*, *node* aktif dalam *linked list*, dan *node* sementara yang akan digunakan dalam pembuatan *node* di *linked list*. Berikan nilai awal *NULL* kepada mereka. Deklarasi node untuk beberapa keperluan, seperti berikut ini:

```
struct tnode *head=NULL, *curr=NULL, *node=NULL;
```

Dengan demikian, sampai saat ini, telah dimiliki tiga *node*. Satu sebagai kepala (*head*), satu sebagai *node* aktif dalam *linked list* (*curr*) dan satu lagi *node* sementara (*node*). Untuk mempermudah pengingatan, ingatlah gambar anak panah yang mengarah ke kanan. Head akan berada pada pangkal anak panah, dan node-node berikutnya akan berbaris ke arah bagian anak panah yang tajam.



Gambar 2. Single Linked List

Apabila diperhatikan, setiap node memiliki petunjuk untuk node sebelahnya. Node terakhir akan diberikan nilai NULL. Dengan demikian, setiap node kebagian jatah.

```
int i;
for (i=0; i<5; i++)
{
node = (struct tnode *)
malloc (sizeof(struct tnode));
node -> data = i;
```

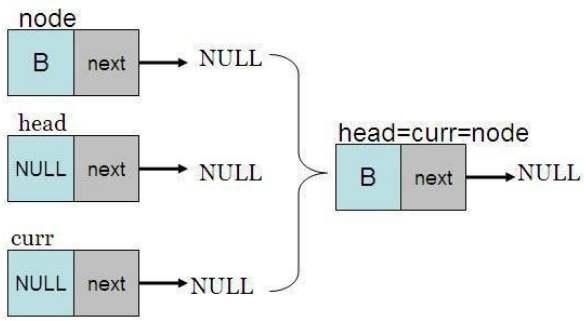
```
if (head == NULL)
{
head = node;
curr = node;
}else
{
curr -> next = node;
curr = node;
}
}
curr -> next = NULL;
```

Berikut adalah penjelasan kode-kode pembuatan singly linked list tersebut. Pertama-tama, akan dibuat perulangan dari 0 sampai 4, yang dimaksudkan untuk membuat lima buah node yang masing-masing field data nya berisikan nilai dari 0 sampai 4. Pembuatan node dilakukan dengan fungsi malloc().

```
for (i=0; i<5; i++)
{
node = (struct tnode *)
malloc (sizeof(struct tnode));
node -> data = i;
...
... }
```

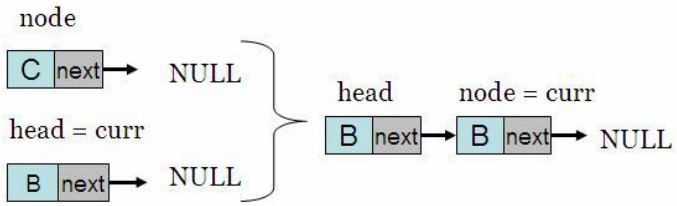
Setelah itu, perlu dibuat node dan penghubung. Pertama-tama, akan diuji apakah head bernilai NULL. Kondisi head bernilai NULL hanya terjadi apabila belum dimiliki satu node pun. Dengan demikian, node tersebut akan dijadikan sebagai head. Node aktif (curr), juga kita dapat dari node tersebut. Kalau head tidak bernilai NULL alias telah dimiliki satu atau lebih node, yang pertama dilakukan adalah menghubungkan pointer next dari node aktif (curr) ke node yang baru saja dibuat. Dengan demikian, baru saja dibuat penghubung antara rantai lama dengan mata rantai baru. Atau, dalam permainan kereta api, pemain paling depan dalam barisan lama akan menempelkan tangannya pada bahu pemain yang baru bergabung. Node aktif (curr) kemudian dipindahkan ke node yang baru dibuat.

```
if (head == NULL)
{
head = node;
curr = node;
}
```



Gambar 3. Membuat elemen pertama SLL

```
else
{
curr->next=node;
curr = node;
}
```



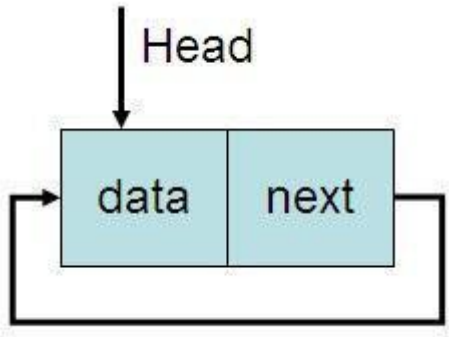
Gambar 4. Penambahan elemen dibelakang SLL

a.2. Single Linked List Circular

Hampir sama dengan single linked list non circular, bahwa dibutuhkan sebuah kait untuk menghubungkan node-node data yang ada, dimana pada node terakhir atau tail yang semula menunjukkan NULL diganti dengan menunjuk ke kepala atau head. Dimana inisialisasi senarai berkait tunggal sirkular menggunakan struc adalah sebagai berikut:

Deklarasi Single Linked List Circular:

```
Struct tnode
{
int data;
tnode *next;
};
void main()
{
head = new tnode;
head->next = head;
}
```

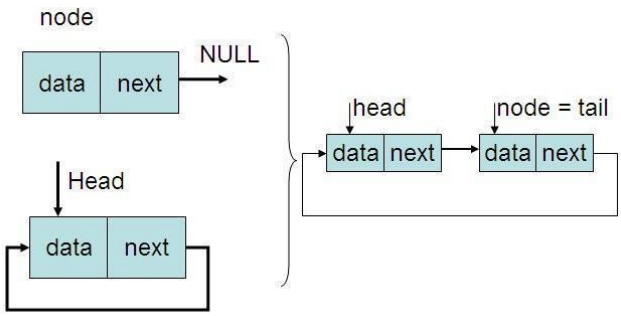


Gambar 5. Single Linked List circular

Menambah node dan membuat tail dari single linked list circular

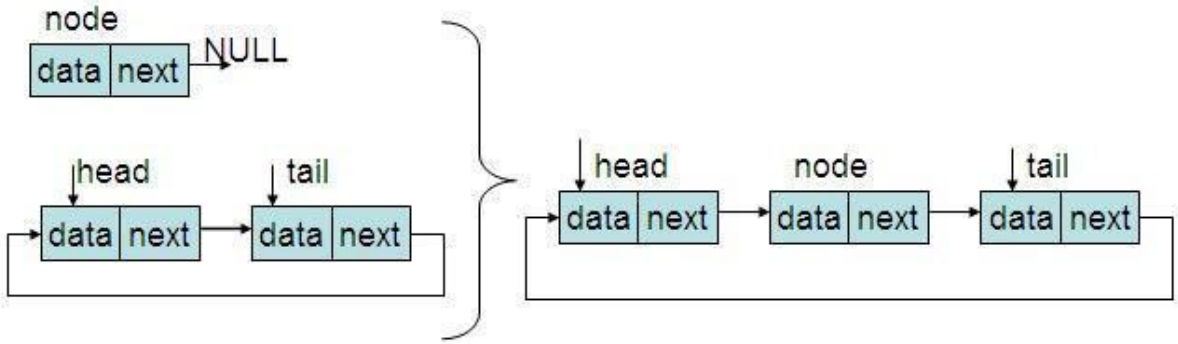
Deklarasi penambahan node baru:

```
Void main()
{
node = new tnode;
tail = new tnode;
node->next = head->next;
head->next = node;
tail = node;
}
```



Gambar 6. Penambahan Node baru

Menyisipkan Node baru

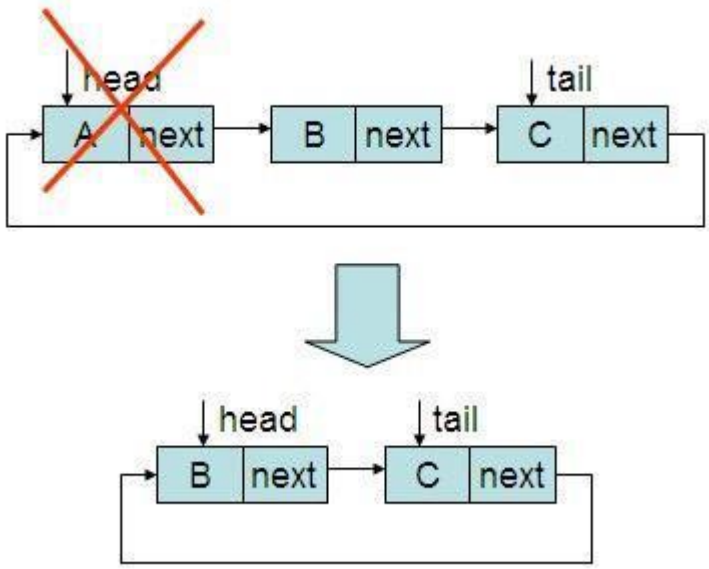


Gambar 7. Menyisipkan Node Baru

Deklarasi menyisipkan node baru menggunakan sintak berikut:

```
Void main()
{
node = new tnode;

node->next = head->next;
head->next = node;
}
```

Menghapus Node dari Single Linked List Circular



Gambar 8. Menghapus node dari SLLC

Deklarasi menghapus node dari single linked list circular, menggunakan sintaks berikut:

```
Void main()
{
hapus = new tnode;
if( head != tail)
{
hapus = head;
head = head->next;
```

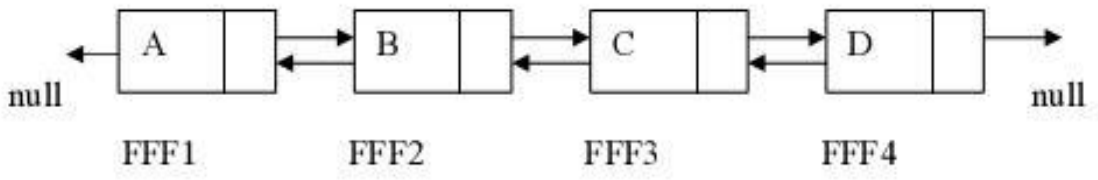
```
tail->next = head;
delete hapus;
}else
{
head = NULL;
tail = NULL;
}
}
```

b. Double Linked List (Senarai berkait ganda)

Double Linked List adalah elemen-elemen yang dihubungkan dengan dua pointer dalam satu elemen dan list dapat melintas baik di depan atau belakang. Elemen double linked list terdiri dari tiga bagian :

1. Bagian data informasi
2. Pointer next yang menunjuk ke elemen berikutnya
3. Pointer prev yang menunjuk ke elemen sebelumnya

b.1. Double Linked List non Circular



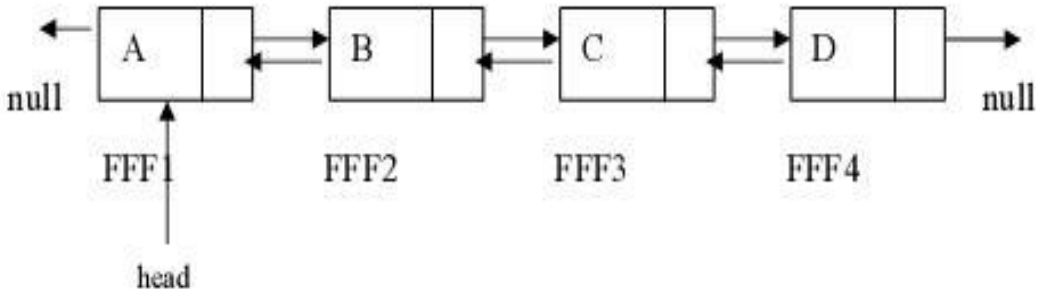
Gambar 9. Ilustrasi *Double Link List Non Circular*

Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya dan ke node sebelumnya. Untuk pembentukan node baru, mulanya pointer next dan prev akan menunjuk ke nilai NULL. Selanjutnya, pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node selanjutnya pada list.

Deklarasi node dibuat dari struct berikut ini :

```
typedef struct TNode{
        int data;
        Tnode *next;
        Tnode *prev;
};
```

Double Linked List Non Circular dengan HEAD membutuhkan satu buah variabel pointer, yaitu: head. Head akan selalu menunjuk pada node pertama.



Gambar 10. Double Linked List Non Circular dengan HEAD

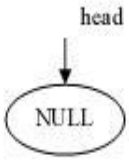
Deklarasi Pointer Penunjuk Kepala Double Linked List

Manipulasi linked list tidak bisa dilakukan langsung ke node yang dituju, melainkan
harus melalui node pertama dalam linked list. Deklarasinya sebagai berikut:

```
TNode *head;
```

Fungsi Inisialisasi Single Linked List non Circular

```
void init(){
–     head = NULL;
}
```



Function untuk mengetahui kosong tidaknya DLLNC

```
int isEmpty(){
if(head == NULL) return 1;
else return 0;
}
```

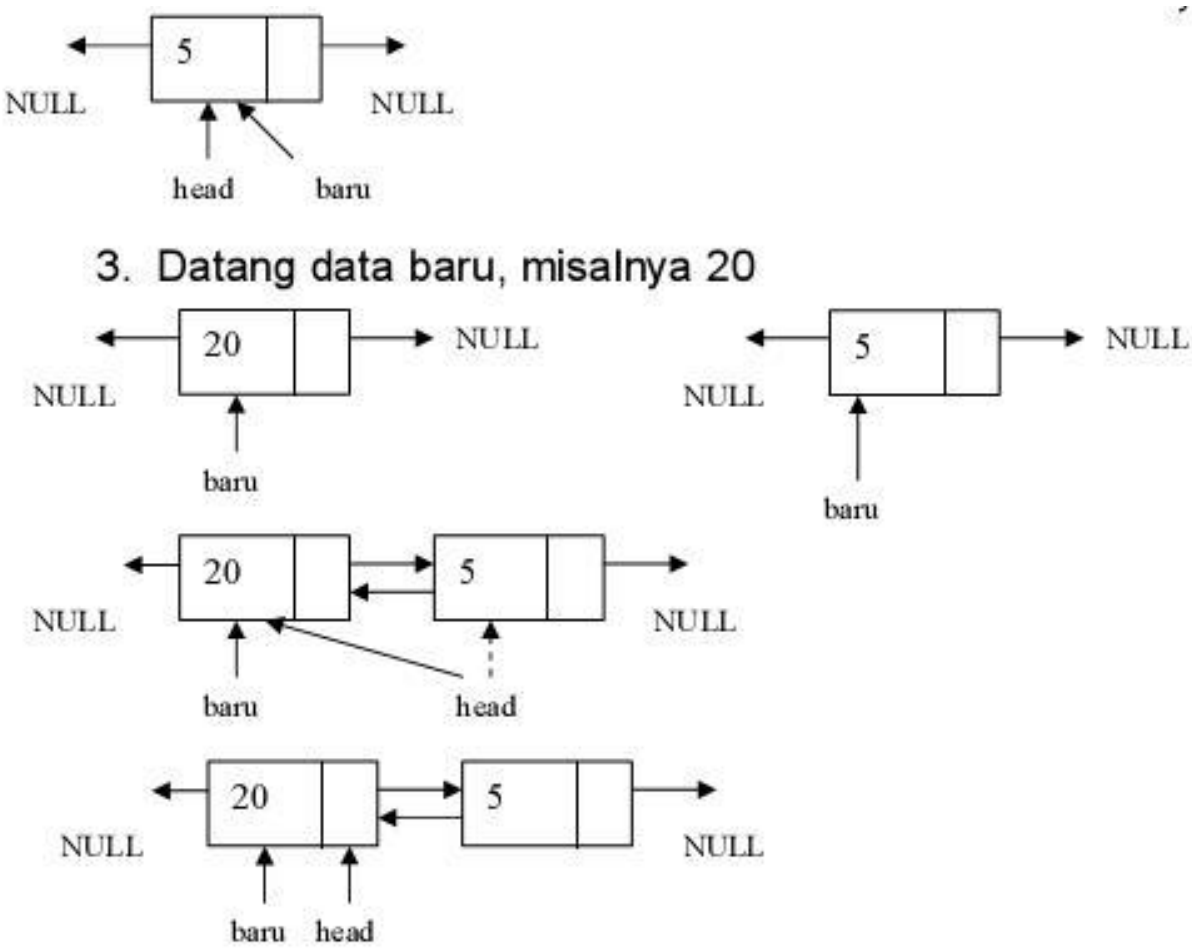
Penambahan data di depan

Penambahan node baru akan dikaitkan di ode paling depan, namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan pada head-nya. Pada prinsipnya adalah mengkaitkan data baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan. Untuk menghubungkan node terakhir dengan node terdepan dibutuhkan pointer bantu.

Fungsi Menambah Di Depan

```
void insertDepan(int databaru){
  TNode *baru;
  baru = new TNode;
  baru->data = databaru;
  baru->next = NULL;
  baru->prev = NULL;
  if(isEmpty()==1){
head=baru;
head->next = NULL;
```

```
head->prev = NULL;
  }
  else {
baru->next = head;
head->prev = baru;
head = baru;
  }
  cout<<"Data masuk\n";
}
```



Gambar11. Ilustrasi Penambahan Node Di Depan

Penambahan data di belakang

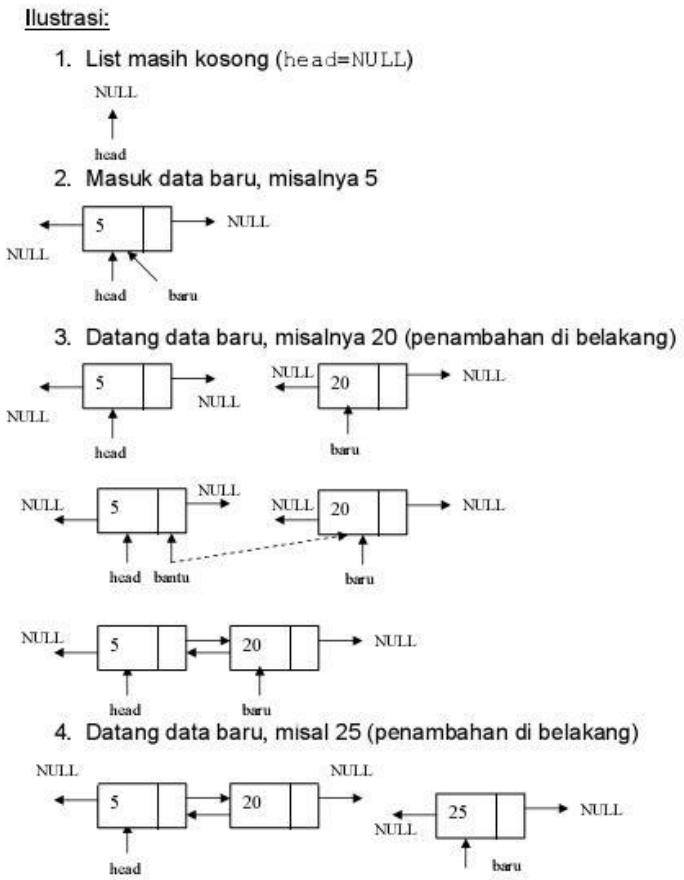
Penambahan data dilakukan di belakang, namun pada saat pertama kali data langsung ditunjuk pada head-nya. Penambahan di belakang lebih sulit karena kita membutuhkan pointer bantu untuk mengetahui data terbelakang, kemudian dikaitkan dengan data baru. Untuk mengetahui data terbelakang perlu digunakan perulangan.

Fungsi penambahan Node ke belakang :

```
Void insertBelakang (int databaru){
      Tnode *baru, *bantu;
      Baru = new Tnode;
```

```
baru->data = databaru;
baru->next = NULL;
baru->prev = NULL;
if(isEmpty()==1){
        head=baru;
        head->next = NULL;
        head->prev = NULL;
}
else{
        bantu=head;
        while(bantu->next!=NULL){
                bantu=bantu->next
        }
        bantu->next = baru;
        baru->prev = bantu;
        }
        Cout<<"Data masuk\n";
        }
```

Ilustrasi Penambahan Node di Belakang



Function untuk menampilkan isi DLLNC

```
void tampil(){
	TNode *bantu;
	bantu = head;
	if(isEmpty()==0){
		while(bantu!=NULL){
		cout<<bantu->data<<" ";
		bantu=bantu->next;
	}
	cout<<endl;
	} else cout<<"Masih kosong\n";
}
```

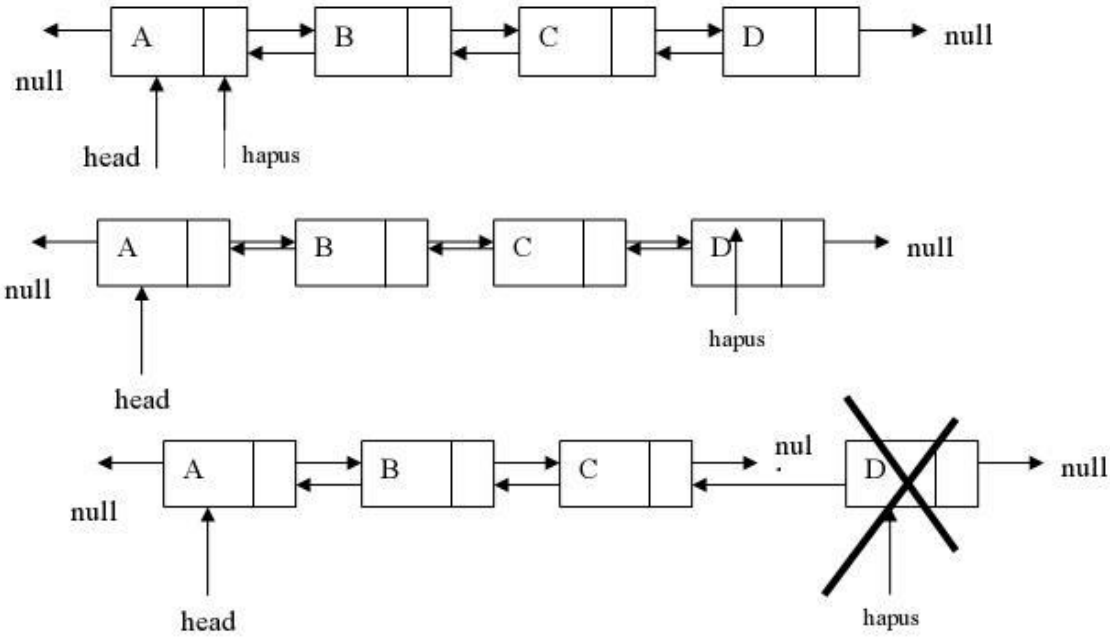
Function untuk menghapus data di depan:

```
void hapusDepan (){
	TNode *hapus;
	int d;
	if (isEmpty()==0){
		if(head->next != NULL){
		   hapus = head;
		   d = hapus->data;
		   head = head->next;
		head->prev = NULL;
		delete hapus;
		} else {
		d = head->data;
		head = NULL;
	}
	cout<<d<<" terhapus\n";
	} else cout<<"Masih kosong\n";
}
```

Fungsi untuk menghapus node terbelakang

```
void hapusBelakang(){
	TNode *hapus;
	int d;
	if (isEmpty()==0){
	   if(head->next != NULL){
		hapus = head;
		while(hapus->next!=NULL){
		hapus = hapus->next;
		}
		d = hapus->data;
		hapus->prev->next = NULL;
		delete hapus;
	   } else {
		d = head->data;
		head = NULL;
	   }cout<<d<<" terhapus\n";
	} else cout<<"Masih kosong\n";
}
```

Ilustrasi Penghapusan Node



Menghapus Node Double Linked List

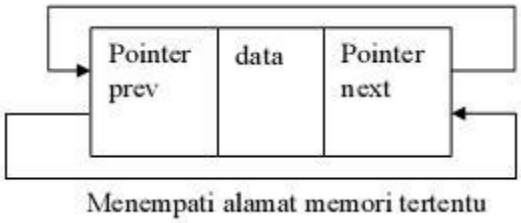
Tidak diperlukan pointer bantu yang mengikuti pointer hapus yang berguna untuk menunjuk ke NULL. Karena pointer hapus sudah bisa menunjuk ke pointer sebelumnya dengan menggunakan elemen prev ke node sebelumnya, yang akan diset agar menunjuk ke NULL setelah penghapusan dilakukan.

Fungsi menghapus semua elemen

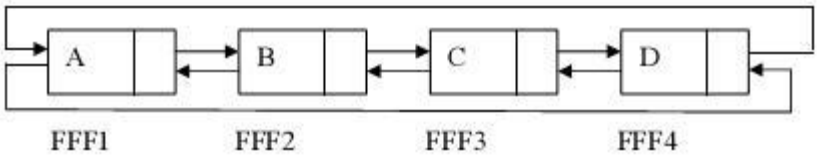
```
void clear(){
      TNode *bantu,*hapus;
      bantu = head;
      while(bantu!=NULL){
         hapus = bantu;
         bantu = bantu->next;
         delete hapus;
   }
   head = NULL;
}
```

b.2. Double Linked List Circular

Double Linked List Circular (DLLC) adalah linked list dengan menggunakan pointer, dimana setiap node memiliki 3 field, yaitu 1 field pointer yang menunjuk pointer berikutnya (next), 1 field menunjuk pointer sebelumnya (prev), serta sebuah field yang berisi data untuk node tersebut dengan pointer next dan pre-nya menunjuk ke dirinya sendiri secara circular.



Ilustrasi DLLC



Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya dan ke node sebelumnya. Untuk pembentukan node baru, mulanya pointer next dan prev akan menunjuk ke dirinya sendiri. Jika sudah lebih dari satu node, maka pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node sesudahnya.

Deklarasi dan node baru DLLC

Deklarasi node

Dibuat dari struct berikut ini:

```
typedef struct TNode{
int data;
TNode *next;
Tnode *prev;
};
```

Pembentukan node baru

Digunakan keyword new yang berarti mempersiapkan sebuah node baru berserta alokasi
memorinya.

```
TNode *baru;
baru = new TNode;
baru->data = databaru;
baru->next = baru;
baru->prev = baru;
```

Penambahan data di depan

Penambahan node baru akan dikaitan di node **paling depan,** namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan pada head nya. Pada prinsipnya adalah mengkaitkan data baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan. Untuk menghubungkan node terakhir dengan node terdepan dibutuhkan pointer bantu.

```
void     insertDepan(int databaru){
  TNode *baru, *bantu;
        baru = new TNode;
```

```
        baru->data = databaru;
        baru->next = baru;
        baru->prev = baru;
    if(isEmpty()==1){
        head=baru;
        head->next = head;
        head->prev = head;
    }
    else {
        bantu = head->prev;
        baru->next = head;
        head->prev = baru;
        head = baru;
        head->prev = bantu;
        bantu->next = head;
    }
    cout<<"Data masuk\n";
}
```

Penambahan data di belakang

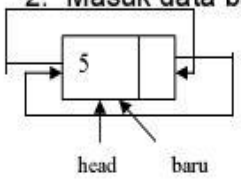
Penambahan data dilakukan **di belakang**, namun pada saat pertama kali data langsung ditunjuk pada head-nya. Penambahan di belakang lebih sulit karena kita membutuhkan pointer bantu untuk mengetahui data terbelakang, kemudian dikaitkan dengan data baru. Untuk mengetahui data terbelakang perlu digunakan perulangan.

```
void insertBelakang (int databaru){
    Tnode *baru, *bantu;
        baru = new Tnode;
        baru->data = databaru;
        baru->next = baru;
        baru->prev = baru;
        if(isEmpty()==1){
            head=baru;
            head->next = head;
            head->prev = head;
        }
        else{
            bantu=head->prev;
            bantu->next = baru;
            baru->prev = bantu;
            baru->next = head;
            head->prev = baru;
        }
        Cout<<”Data masuk\n”;
    }
```

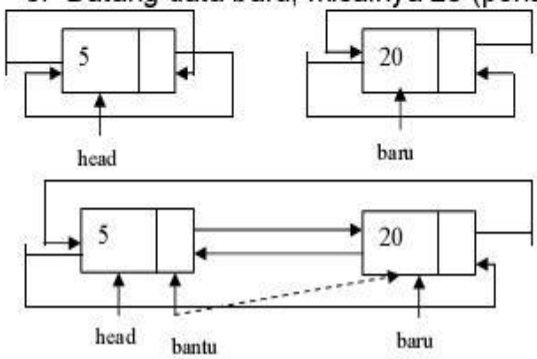
Ilustrasi:

1. List masih kosong (`head=NULL`)

2. Masuk data baru, misalnya 5

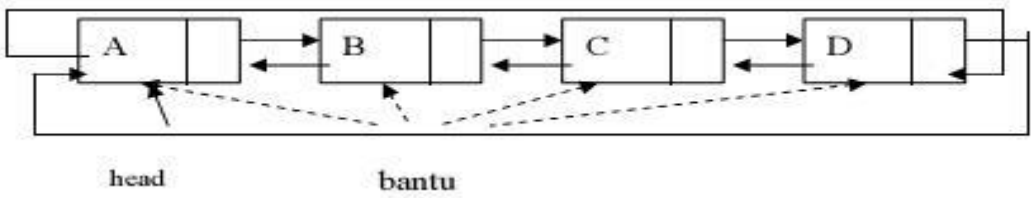


3. Datang data baru, misalnya 20 (penambahan di belakang)



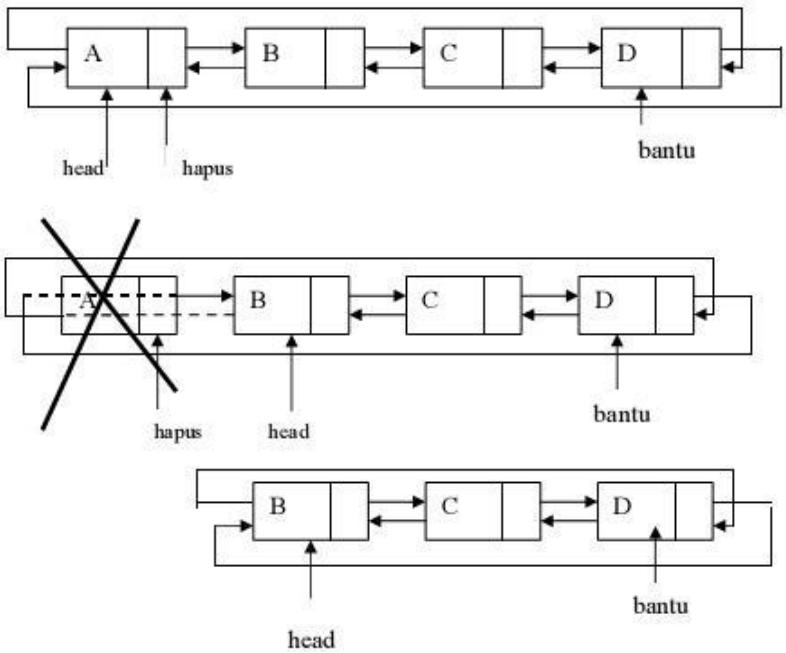
Function untuk menampilkan isi linked list

```
void tampil(){
        TNode *bantu;
        bantu = head;
        if(isEmpty()==0){
                do{
                        cout<<bantu->data<<" ";
                        bantu=bantu->next;
                }while(bantu!=head);
                cout<<endl;
        } else cout<<"Masih kosong\n";
}
```



Function untuk menghapus node terbelakang

```
void   hapusDepan (){
       TNode *hapus,*bantu;
       int d;
       if (isEmpty()==0){
         if(head->next != head){
                hapus = head;
                d = hapus->data;
                bantu = head->prev;
                head = head->next;
                bantu->next = head;
                head->prev = bantu;
                delete hapus;
       } else {
                d = head->data;
                head = NULL;
       }
       cout<<d<<" terhapus\n";
       } else cout<<"Masih kosong\n";
}
```



Function untuk menghapus node terbelakang

```
void   hapusBelakang(){
       TNode *hapus,*bantu;
       int d;
       if (isEmpty()==0){
         if(head->next != head){
                bantu = head;
                while(bantu->next->next != head){
                        bantu = bantu->next;
                }
```

```
                hapus = bantu->next;
                d = hapus->data;
                bantu->next = head;
                delete hapus;
        } else {
                d = head->data;
                head = NULL;
        }
        cout<<d<<" terhapus\n";
        } else cout<<"Masih kosong\n";
}
```

Function untuk menghapus semua elemen

```
void   clear(){
        TNode *bantu,*hapus;
        if (isEmpty()==0){
                bantu = head;
                while(bantu->next!=head){
                hapus = bantu;
                bantu = bantu->next;
                delete hapus;
                }
                head = NULL;
        }
}
```

Latihan

```
#include <iostream>
#include <conio.h>
#include <iomanip> //setw()
using namespace std;

struct node
{
    int data;
    node* next; //untuk menghubungkan dengan node lain, tipe data dibuat sm sprt aturan penggunaan pointer
};

node* head;
node* tail;
node* curr;
node* entry;
node* del;

void inisialisasi()
{
    head = NULL;
    tail = NULL;
}

void input(int dt)
{
    entry = (node* )malloc(sizeof(node)); //alokasi memori
    entry->data = dt;
    entry->next = NULL;
    if(head==NULL)
    {
        head = entry;
        tail = head;
    }
    else
    {
        tail->next = entry;
        tail = entry;
    }
}

void hapus()
{
    int simpan;
    if(head==NULL)
    {
        cout<<"\nlinked list kosong, penghapusan tidak bisa dilakukan"<<endl;
    }
    else
    {
        simpan = head->data;
        cout<<"\ndata yang dihapus adalah"<<simpan<<endl;
        //hapus depan
        del = head;
        head = head->next;
        delete del;
    }
}

void cetak()
{
    curr = head;
    if(head == NULL)
```

```
63          cout<<"\ntidak ada data dalam linked list"<<endl;
64      else
65      {
66          cout<<"\nData yang ada dalam linked list adalah"<<endl;
67          cout<<setw(6);
68          while(curr!=NULL)
69          {
70              cout<<curr->data<<"->";
71              curr = curr->next;
72          }
73          cout<<endl;
74      }
75  }
76
77  void menu()
78  {
79      char pilih, ulang;
80      int data;
81
82      do
83      {
84          system("cls");
85          cout<<"SINGLE LINKED LIST NON CIRCULAR"<<endl;
86          cout<<"--------------------------------"<<endl;
87          cout<<"Menu : "<<endl;
88          cout<<"1. Input data"<<endl;
89          cout<<"2. Hapus data"<<endl;
90          cout<<"3. Cetak data"<<endl;
91          cout<<"4. Exit"<<endl;
92          cout<<"Masukkan pilihan Anda :";
93          cin>>pilih;
94          switch(pilih)
95          {
96          case '1':
97              cout<<"\nMasukkan data :";
98              cin>>data;
99              input(data);
100             break;
101         case '2':
102             hapus();
103             break;
104         case '3':
105             cetak();
106             break;
107         case '4':
108             exit(0);
109             break;
110         default :
111             cout<<"\nPilih ulang"<<endl;
112         }
113         cout<<"Kembali ke menu?(y/n)";
114         cin>>ulang;
115         }while(ulang=='y' || ulang=='Y');
116     }
117
118 int main()
119     {
120         inisialisasi();
121         menu();
122
123         return EXIT_SUCCESS;
124     }
```

Tugas Praktikum

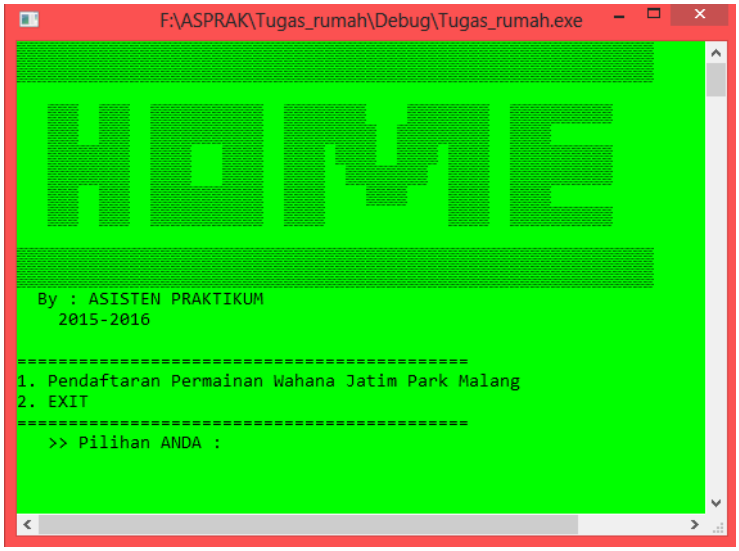
TROUBLESHOOT PROGRAM :

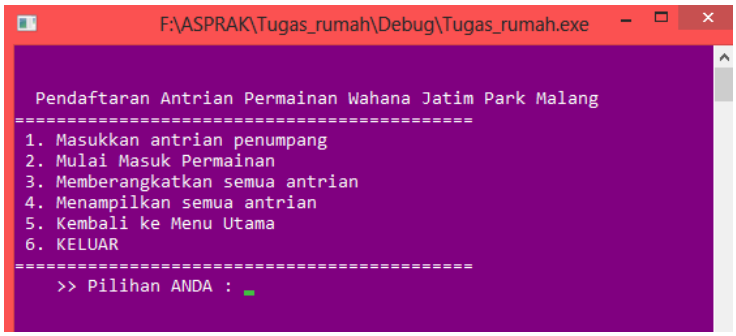
```
#include <stdio.h>
#include <stdlib.h>

//definisikan struct
struct SNode{
    int data;
    struct SNode *next;
    struct SNode *before;
};

struct SNode *n_awal,   //node awal dari linked list
    *n_tampil,          //node bantu untuk tampilkan data
    *n_bantu;           //node bantu untuk insert node baru

int iterasi

//inisialisasi linked list
void inisialisasi(){
    n_awal = NULL;
}

//tampilkan linked list
void tampil_list(){
    n_tampil = n_awal

    while(n_tampil !== NULL){
        printf("%d", n_tampil->data);
        n_tampil = n_tampil->before;
    }
printf("\n");
}

//method untuk insert node baru
void insert_node(int data){

    //node baru yang akan di insert
    struct SNode *n_insert;
    n_insert = (struct SNode *)malloc(sizeof(struct SNode));
    n_insert->data = iterasi;
    n_insert->next = NULL;
    n_insert->before = NULL;

    //kondisi linked list masih kosong
    if(n_awal == NULL){
        n_awal = n_insert;
    }else{
        n_bantu = n_awal;

        //cari node terakhir
        while(n_bantu->next != NULL){
            n_bantu = n_bantu->next;
        }

        //hubungkan node terakhir dengan node baru
        if(n_bantu->next==NULL){
        {
            n_bantu->next = n_insert;
            insert->before = n_bantu;
        }
    }
}
```

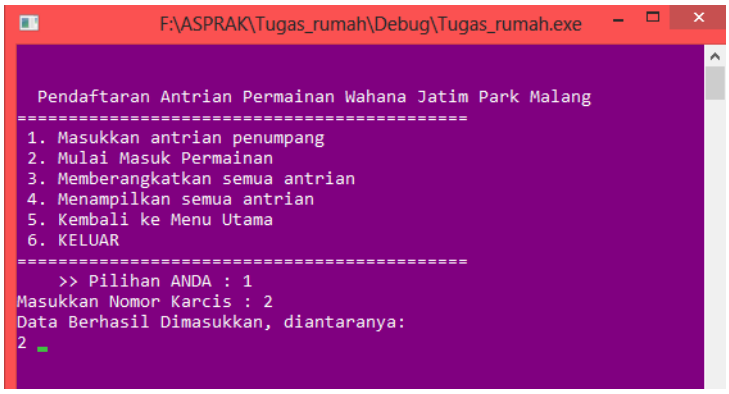
```
63  int main(int argc, char *argv[])
64  {
65      inisialisasi();
66      for(iterasi = 11; iterasi<=15; iterasi++)
67      {
68          insert_node(iterasi);
69      }
70
71      tampil_list();
72
73      system("PAUSE");
74      return 0;
75  }
```

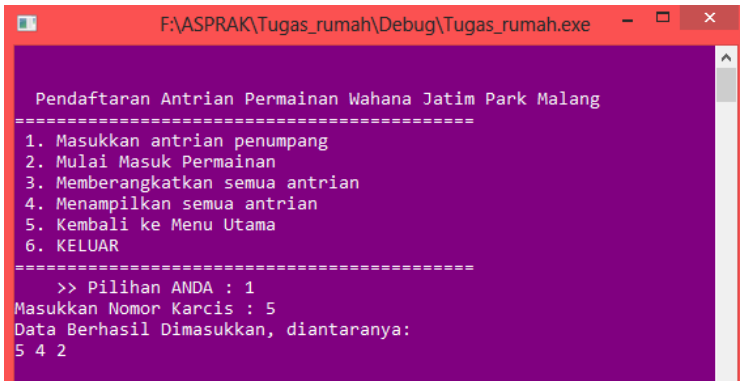
TUGAS RUMAH

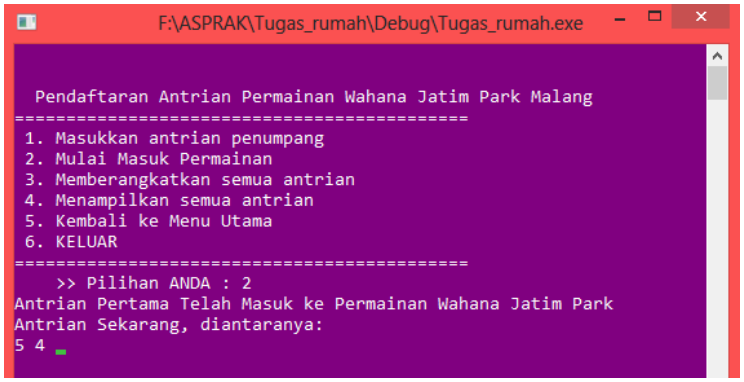
Buatlah program seperti berikut ini menggunakan metode Linked List :

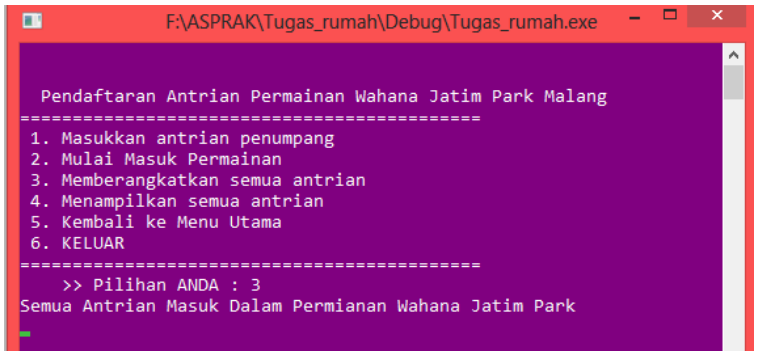





F:\ASPRAK\Tugas_rumah\Debug\Tugas_rumah.exe
Pendaftaran Antrian Permainan Wahana Jatim Park Malang
1. Masukkan antrian penumpang
2. Mulai Masuk Permainan
3. Memberangkatkan semua antrian
4. Menampilkan semua antrian
5. Kembali ke Menu Utama
6. KELUAR
>> Pilihan ANDA : 1
Masukkan Nomor Karcis : 2
Data Berhasil Dimasukkan, diantaranya:
2


F:\ASPRAK\Tugas_rumah\Debug\Tugas_rumah.exe
Pendaftaran Antrian Permainan Wahana Jatim Park Malang
1. Masukkan antrian penumpang
2. Mulai Masuk Permainan
3. Memberangkatkan semua antrian
4. Menampilkan semua antrian
5. Kembali ke Menu Utama
6. KELUAR
>> Pilihan ANDA : 1
Masukkan Nomor Karcis : 5
Data Berhasil Dimasukkan, diantaranya:
5 4 2


F:\ASPRAK\Tugas_rumah\Debug\Tugas_rumah.exe
Pendaftaran Antrian Permainan Wahana Jatim Park Malang
1. Masukkan antrian penumpang
2. Mulai Masuk Permainan
3. Memberangkatkan semua antrian
4. Menampilkan semua antrian
5. Kembali ke Menu Utama
6. KELUAR
>> Pilihan ANDA : 2
Antrian Pertama Telah Masuk ke Permainan Wahana Jatim Park
Antrian Sekarang, diantaranya:
5 4


F:\ASPRAK\Tugas_rumah\Debug\Tugas_rumah.exe
Pendaftaran Antrian Permainan Wahana Jatim Park Malang
1. Masukkan antrian penumpang
2. Mulai Masuk Permainan
3. Memberangkatkan semua antrian
4. Menampilkan semua antrian
5. Kembali ke Menu Utama
6. KELUAR
>> Pilihan ANDA : 3
Semua Antrian Masuk Dalam Permianan Wahana Jatim Park