**MAKALAH**

**SENARAI BERANTAI**



DOSEN PENGAJAR

Indrawati, SST.MT

**DISUSUN OLEH**

DIKI CANDRA

**NIM. 2022903430010**

**Kelas : TRKJ-1B**

**POLITEKNIK NEGERI LHOKSEUMAWE**

**JURUSAN TEKNOLOGI KOMUNIKASI DAN KOMPUTER**

**PRODI D-IV TEKNOLOGI REKAYASA KOMPUTER DAN JARINGAN**

**2022/2023**

# BAB I

PENDAHULUAN

## 1.1 Latar Belakang

Pada array, apabila programmer ingin menyimpan data, programmer diharuskan untuk mendefinisikan besar array terlebih dahulu, seringkali programmer mengalokasikan array yang sangat besar(misal 100). Hal ini tidak efektif karena seringkali yang dipakai tidak sebesar itu. Dan apabila programmer ingin menyimpan data lebih dari seratus data, maka hal itu tidak dapat dimungkinkan karena sifat array yang besarnya statik. Linked list adalah salah satu struktur data yang mampu menutupi kelemahan tersebut.

Linked list (list bertaut) adalah salah satu struktur data dasar yang sangat fundamental dalam bidang ilmu komputer. Dengan menggunakan linked list maka programmer dapat menimpan datanya kapanpun dibutuhkan. Linked list mirip dangan array, kecuali pada linked list data yang ingin disimpan dapat dialokasikan secara dinamis pada saat pengoperasian program (run-time).

## 1.2 Tujuan

1. Mengetahui pengertian linked list.

2. Mengetahui macam-macam dari linked list.

3. Mengetahui contoh program sederhana yang menggunakan linked list.

BAB II

ISI

## **2.1 Pengertian**

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 simpull dapat berbentuk suatu struktur atau class. Simpul harus mempunyai satu atau lebih elemen struktur atau class yang berisi data.

Senarai berkait (linked list) adalah salah satu struktur data dasar yang sangat fundamental dalam bidang ilmu komputer. Dengan menggunakan linked list maka programmer dapat menimpan datanya kapanpun dibutuhkan. Linked list mirip dangan array, kecuali pada linked list data yang ingin disimpan dapat dialokasikan secara dinamis pada saat pengoperasian program (run-time).

Didalam banyak aplikasi, ukuran dari data tidak diketahio saat kompile, hal ini bisa merupakan suatu atribut yang baik juga. Setiap node akan berbentuk struct dan memiliki satu buah field yang bertipe struct yang sama, yang berfungsi sebagai pointer. Dalam menghubungkan setiap node, kita dapat menggunakan cara first-create-first-access maupun first-create-last-access.

Linked list saling terhubung dengan bantuan variabel pointer. Masing-masing data dalam linked list disebut dengan node (simpul) yang menempati alokasi memori secara dinamis dan biasanya berupa struct yang terdiri dari beberapa field.

**Istilah-istilah**

1. Simpul Simpul terdiri dari dua bagian, yaitu : ·
   * Bagian data ·
   * Bagian pointer yang menunjuk ke simpul berikutnya.
2. First Variable first berisi alamat/pointer menunjuk lokasi simpul pertama linked list, digunakan sebagai awal penelusuran linked list.
3. Nil atau null Tidak bernilai, untuk menyatakan tidak mengacu ke manapun.
4. Simpul terakhir Simpul terakhir linked list berarti tidak menunjuk simpul berikutnya. Tidak terdapat alamat disimpan di field pointer (bagian kedua dari simpul). Nilai null atau nil disimpan di field pointer pada simpul terakhir.

Maka

1) Linked list kosong adalah linked list dengan First = Nil

2) Elemen terakhir linked list dikenal dengan last^.next = Nil

Cara melinierkan urutan adalah dengan menggunakan Penuding atau Pointer. Setiap simpul terdiri atas dua bagian yaitu :

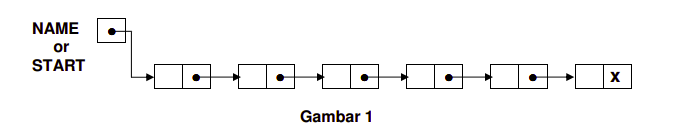
1. Berisi informasi data

2. Merupakan field link atau nextpointer.

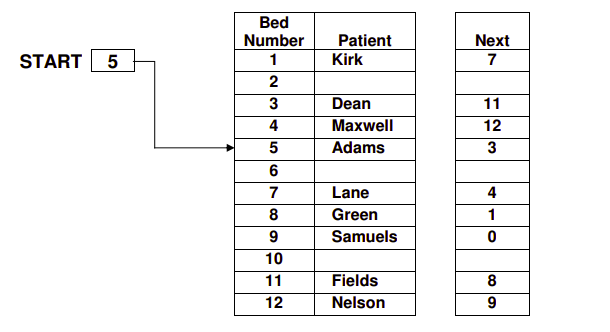
Link menghubungkan satu elemen data ke elemen data lainnya, sehingga urutan elemen data tersebut membentuk suatu linier list.

Link akan bernilai = 0 bila tidak menuding ke data (simpul) lainnya. Penuding ini disebut Penuding Nol.

**Gambar list dengan 6 simpul**



Contoh : Pada bangsal sebuah rumah sakit terdapat 12 tempat tidur. Sembilan di antaranya telah ditempati Pasien. Kita hendak membuat list nama para pasien tersebut secara alfabetik.



**PENYAJIAN LINKED LIST DALAM MEMORI**

Disajikan dalam bentuk sebagai berikut :

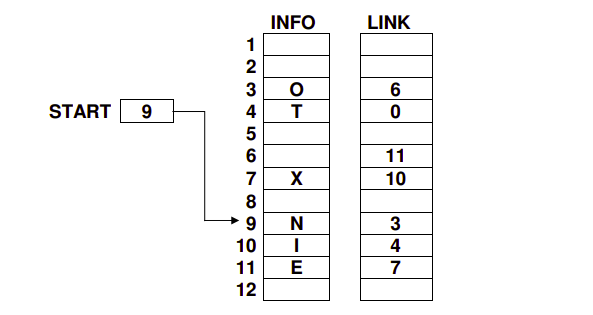
1. Array INFO(K) : menyajikan bagian informasi

2. Array LINK(K) : field nextpointer

3. Variabel START : untuk menyimpan alamat dari elemen LIST Pada bagian akhir dari LIST, nextpointer bernilai NULL.

Contoh 1:

Menggambarkan suatu linked list dalam memori. Data dalam Array INFO(K) adalah sebuah karakter tunggal.



String yang dinyatakan dari contoh tersebut adalah NO EXIT.

Di sini :

START = 9 INFO[9] = N, elemen pertama adalah N

LINK(9) = 3 INFO[3] = O, elemen kedua adalah O

LINK(3) = 6 INFO[6] = Blank, elemen ketiga adalah blank

LINK(6) = 11 INFO[11] = E, elemen keempat adalah E

LINK(11) = 7 INFO[7] = X, elemen kelima adalah X

LINK(7) = 10 INFO(10] = I, elemen keenam adalah I

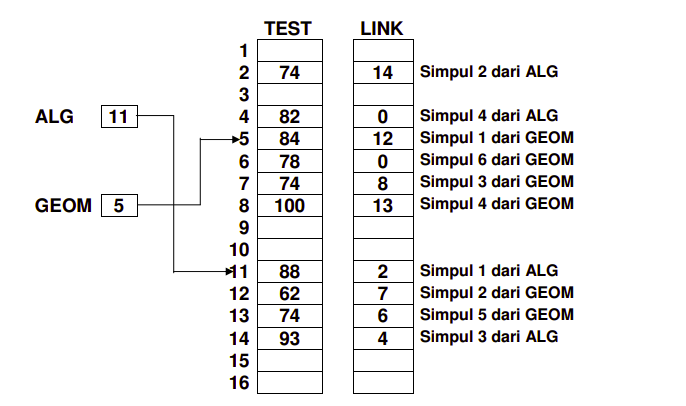
LINK(10) = 4 INFO[4] = T, elemen ketujuh adalah T

LINK(4) = 0 Null,

list terakhir

Contoh 2 :

Memperlihatkan dua buah linked list, ALG dan GEOM, yang berturutturut berisi nilai matakuliah Algoritma dan Geometri, dapat tersimpan bersama dalam array TEST dan LINK yang sama. Pointer ALG berisi nilai 11, yakni lokasi dari simpul pertama list ALG, Pointer GEOM berisi nilai 5, yakni lokasi simpul pertama dari list GEOM.



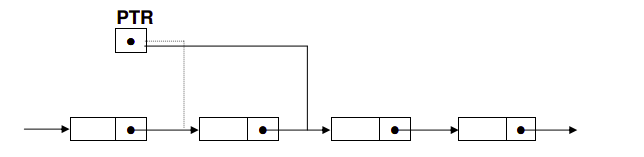
Mengikuti pointer tersebut, maka :

Nilai dari ALG berturut-turut adalah : 88, 74, 93, 82

Nilai dari GEOM berturut-turut adalah : 84, 62, 74, 100, 74, 78

**KUNJUNGAN LINKED LIST**

Traversal atau kunjungan simpul list sesuai urutan untuk memproses setiap simpul tepat satu kali. Algoritma traversal, menggunakan variabel penuding PTR untuk menuding simpul yang sedang di proses saat ini. Karena itu LINK(PTR) akan menuding simpul berikut dalam list. Statement PTR := LINK(PTR) akan menggerakkan penuding ke simpul berikutnya.



Algoritma Traversal secara rinci

Mula-mula, kita awali dengan memberi nilai kepada PTR, sama dengan START. Kita proses INFO(PTR), yakni informasi pada simpul pertama dalam List. Selanjutnya PTR diperbaharui melalui statement PTR := LINK(PTR). Sekarang proses INFO(PTR), yakni informasi pada simpul kedua. Demikian seterusnya sampai nilai PTR = NULL, akhir dari traversal.

**ALGORITMA**

1. PTR := START.

2. Kerjakan Langkah 3 dan 4 dalam hal PTR <> NULL :

3. Proses INFO(PTR).

4. PTR := LINK(PTR).

5. EXIT.

**CARI (SEARCHING) DALAM LINKED LIST**

1. Cari dalam list tidak terurut

2. Cari dalam list terurut

**CARI DALAM LIST TIDAK TERURUT**

Melakukan Traversal Simpul list, sambil setiap kali memeriksa apakah informasi dalam simpul yang tengah dikunjungi tersebut sama dengan ITEM.

Dalam algoritma ini diperlukan 2 buah pemeriksaan pada setiap putaran yaitu :

1. Memeriksa apakah telah sampai pada akhir dari list, yakni dengan memeriksa apakah PTR = NULL.
2. Memeriksa apakah ITEM telah ditemukan lokasinya, yakni dengan memeriksa apakah INFO(PTR) = ITEM

**ALGORITMA**

SEARCH(INFO, LINK, START, ITEM, LOC)

1. PTR := START.

2. Kerjakan langkah 3 dalam hal PTR <> NULL :

3. Jika INFO(PTR) = ITEM, maka : LOC := PTR, exit. Bila tidak PTR := LINK(PTR).

4. LOC := NULL. (Pencarian gagal)

5. Exit.

**CARI DALAM LIST TERURUT**

Melakukan traversal simpul list, sambil setiap kali memeriksa apakah informasi dalam simpul yang tengah dikunjungi tersebut sama dengan ITEM. Karena terurutnya list, tidak perlu melakukan traversal sampai akhir dari list, walau ITEM tidak terdapat dalam list. Begitu INFO(PTR) > ITEM, hentikan proses pencarian. Dalam algoritma ini diperlukan 2 buah pemeriksaan pada setiap putaran yaitu :

1. Memeriksa apakah INFO(PTR) sudah lebih besar dari ITEM, berarti ITEM tidak terdapat dalam list, hentikan proses cari.
2. Memeriksa apakah ITEM telah ditemukan lokasinya, yakni dengan memeriksa apakah INFO(PTR) = ITEM.

**ALGORITMA**

SRCHSL(INFO, LINK, START, ITEM, LOC)

1. PTR := START.
2. Kerjakan langkah 3 dalam hal PTR <> NULL :
3. Jika INFO(PTR) < ITEM, maka : PTR := LINK(PTR). Bila tidak jika ITEM = INFO(PTR), maka : LOC := PTR, dan exit. (pencarian sukses) Bila tidak : LOC := NULL, and exit.
4. LOC := NULL.
5. Exit.

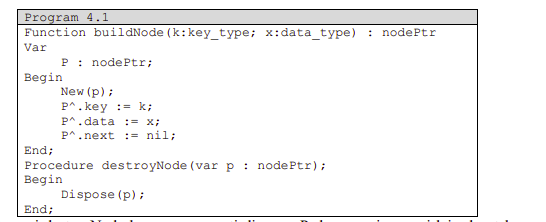
**ALOKASI MEMORI : KOLEKSI SAMPAH**

Ketika menyimpan linked list dalam memori, diasumsikan bahwa selalu dapat dilakukan penyisipan simpul baru ke dalam list, serta penghapusan simpul dari list.

Untuk itu diperlukan suatu mekanisme guna menyediakan memori bagi simpul baru, atau untuk mengelola memori yang sementara ini tidak berguna karena adanya penghapusan simpul, untuk sewaktuwaktu dapat dipakai lagi.

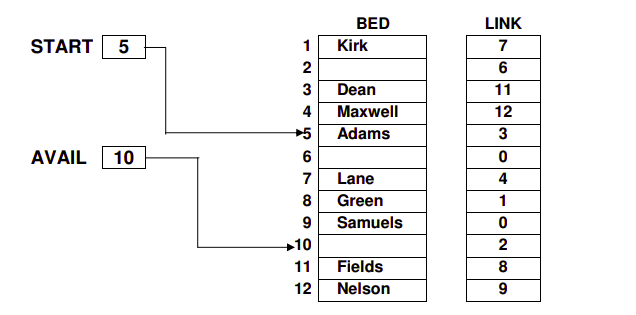
Sel memori dalam array yang tak digunakan, dihimpun menjadi sebuah linked list lain yang menggunakan variabel penuding list berupa array AVAIL, dituliskan sebagai :

LIST(INFO, LINK, START, AVAIL)



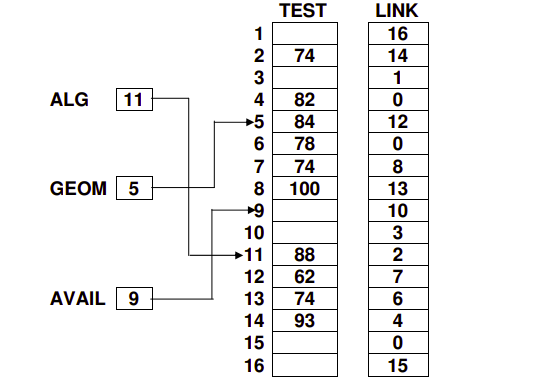
Contoh :

Pandang Daftar Pasien pada contoh yang lalu. Daftar kita simpan dalam dua array BED dan LINK. Pasien tempat tidur K dinyatakan sebagai BED(K). Maka ruang yang tersedia dalam array BED tersebut dapat dikaitkan seperti gambar di bawah ini.



Contoh :

Perhatikan bahwa masing-masing list ALG dan GEOM boleh menggunakan list AVAIL. Dapat dicatat bahwa AVAIL = 9, maka TEST(9) adalah simpul bebas pertama dalam list AVAIL tersebut. Karena LINK(AVAIL) = LINK(9) = 10, maka TEST(10) adalah simpul bebas kedua dalam AVAIL, demikian seterusnya.

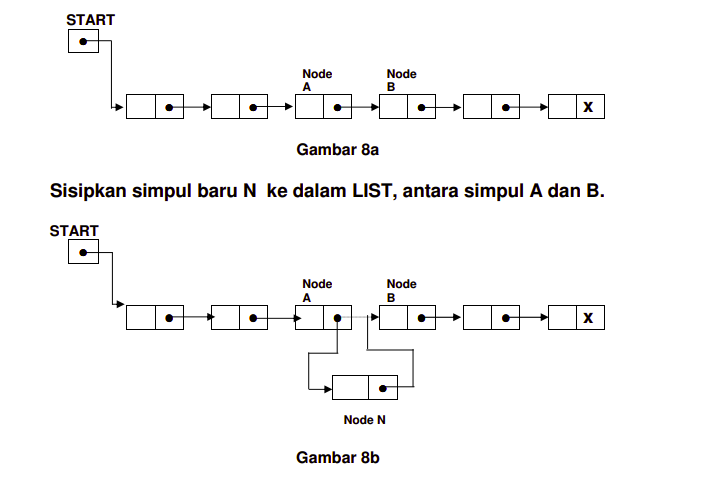


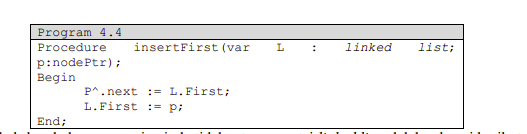
**KOLEKSI SAMPAH**

Koleksi Sampah atau Garbage Collection adalah suatu metode dengan sistem operasi dapat secara periodik mengumpulkan semua ruang akibat penghapusan simpul list tersebut ke dalam list ruang bebas.

**PENYISIPAN SIMPUL KE DALAM LINKED LIST**

Misalkan LIST adalah linked list dengan A dan B adalah dua simpul yang berurutan.

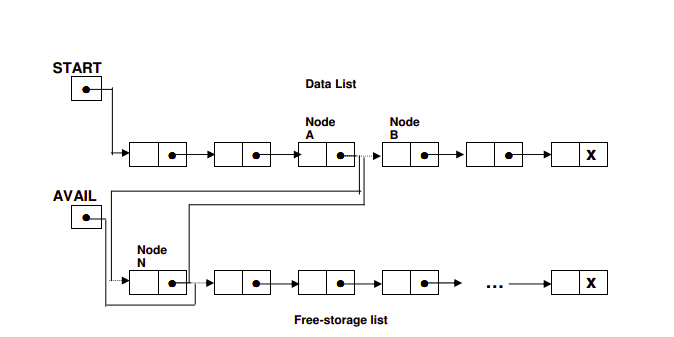




Di sini simpul A sekarang menuding ke simpul baru N, dan simpul N menuding ke simpul B, yang tadinya dituding oleh A. Pandang bahwa linked list tersimpan di dalam memori dalam bentuk

LIST(INFO, LINK, START, AVAIL)

Gambar 8b, tidak dapat memperlihatkan bahwa simpul baru N memanfaatkan ruang memori dari list AVAIL. Untuk kemudahan proses, simpul pertama AVAIL dipakai untuk menyimpan simpul baru N tersebut.



Perhatikan bahwa field Penuding berubah sebagai berikut :

1. Field nextpointer dari simpul A sekarang menuding ke simpul baru N, terhadap mana sebelumnya AVAIL menuding.
2. AVAIL sekarang menuding ke simpul kedua pada ruang bebas, terhadap mana sebelumnya simpul N menuding.
3. .Field nextpointer dari simpul N, sekarang menuding ke simpul B, yang tadinya dituding oleh simpul A.

Di sini terdapat 2 kasus khusus yaitu :

1. Jika simpul baru N adalah simpul pertama dalam list, maka START akan menuding N.

2. Jika simpul baru N adalah simpul terakhir dalam list, maka N akan berisi penuding nol.

**Contoh :**

Pandang Daftar Alfabetik Pasien. Misalkan seorang pasien baru Hughes masuk. Perhatikan bahwa :

1. Hughes ditempatkan di ranjang 10, yakni ranjang pertama yang kosong (tersedia).

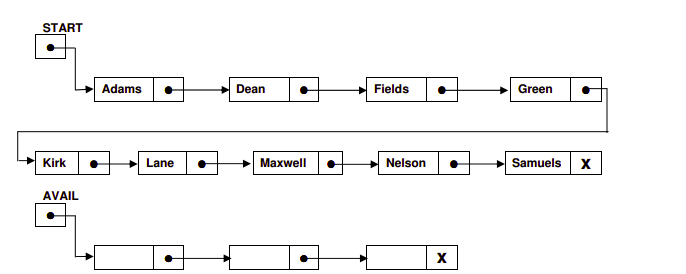
2. Hughes akan disisipkan dalam list, antara Green dan Kirk.

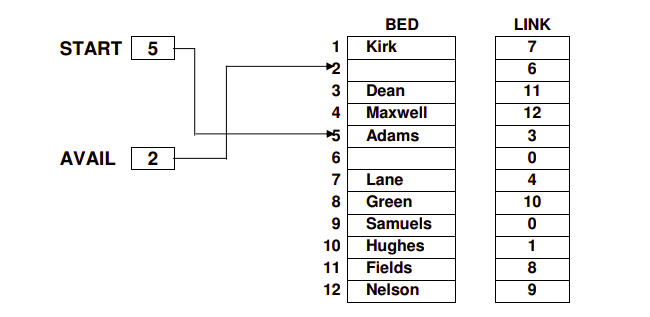
Terjadi 3 perubahan dalam Field Penuding sebagi berikut :

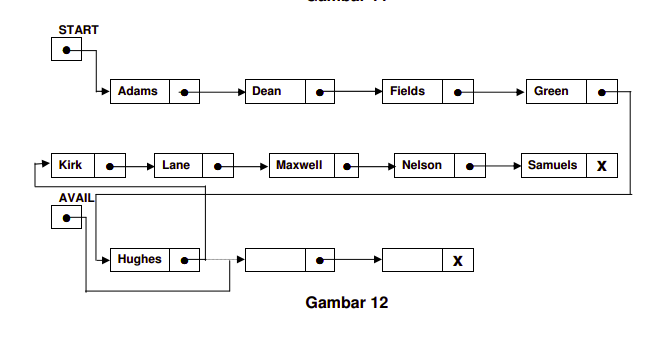
LINK(8) = 10 (sekarang Green menuding Hughes)

LINK(10) = 1 (sekarang Hughes menuding Kirk)

AVAIL = 2 (sekarang AVAIL menuding ranjang kosong kedua)



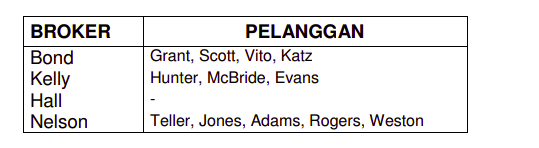


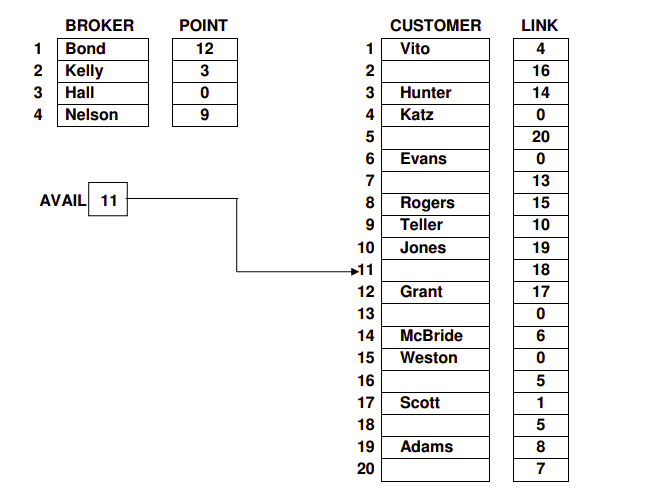


Contoh :

Pandang suatu agen penjualan yang mempunyai 4 orang broker (perantara). Setiap broker mempunyai list customer (pelanggan) masing-masing. Di sini keempat list pelanggan digabung dalam satu linked list dengan array CUSTOMER berisi nama pelanggan dan array LINK merupakan nextpointer. Nama broker ditempatkan dalam array BROKER, dan digunakan pula variabel penuding dalam bentuk array POINT, sedemikian sehingga POINT(K) menuding ke lokasi dari simpul pertama BROKER(K).

Dapat kita lihat bahwa :





Pandang daftar broker dan pelanggannya. Karena list pelanggan tidak terurut, asumsikan bahwa setiap pelanggan baru ditambahkan pada awal list.

Misalkan Gordan adalah pelanggan baru dari Kelly. Perhatikan bahwa :

1. Gordan dimasukkan sebagai CUSTOMER(11), yakni simpul tersedia pertama.

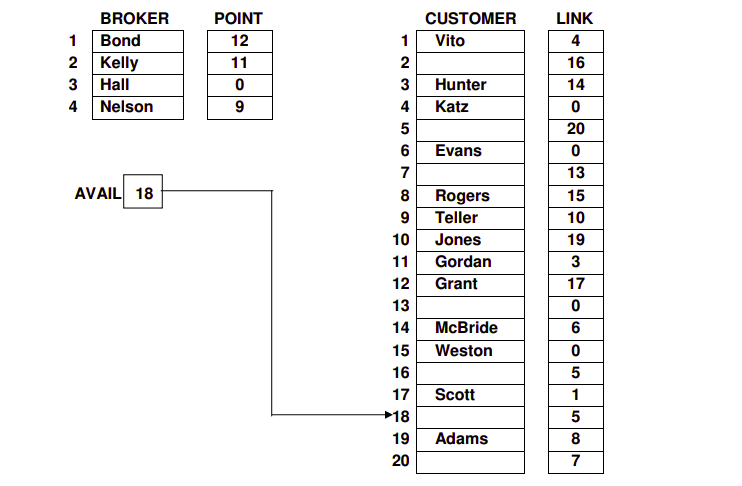
2. Gordan disisipkan di muka Hunter, yang tadinya adalah pelanggan pertama dari Kelly.

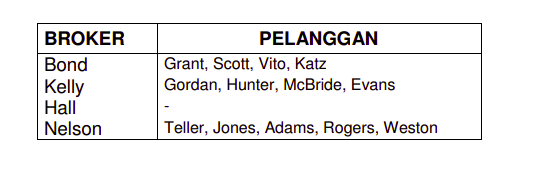
Di sini terjadi 3 perubahan dalam Field Penuding sebagai berikut :

POINT(2) = 11 (sekarang list mulai dengan Gordan) P

OINT(11) = 3 (Gordan menuding Hunter)

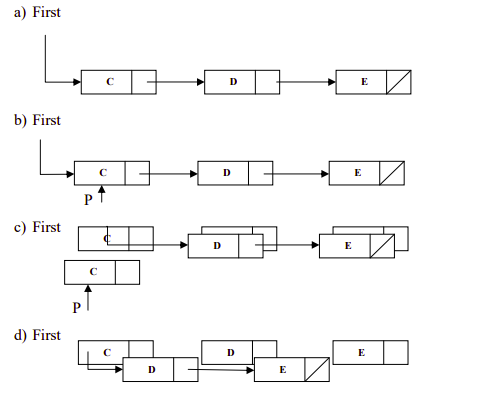
AVAIL = 18 (sekarang AVAIL menuding ke simpul tersedia berikutnya)





**Penghapusan simpul**

Penghapusan simpul pertama Operasi ini akan menghapus elemen pertama linked list. Sebelum memanggil prosedur ini harus dijamin linked list tidak kosong. Skema penghapusan ini digambarkan sebagai berikut:

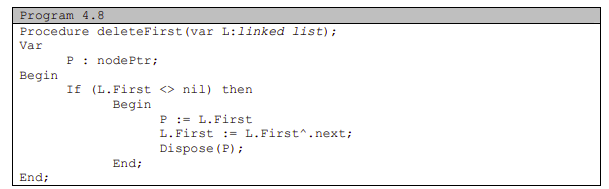


Langkah-langkah yang menjamin ketidakputusan rantai linked list adalah sebagai berikut:

1. Element First dicatat di suatu elemen

2. L.First mencatat elemen selanjutnya setelah L.First

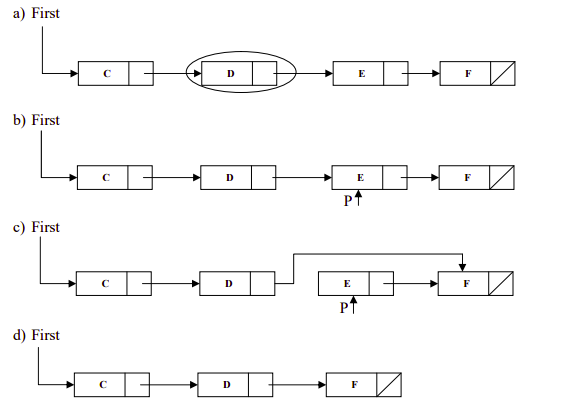
Kedua langkah tersebut mengisolasi elemen pertama sebelumnya.



Prosedur Dispose merupakan prosedur standar di Pascal yang akan mengembalikan elemen dinamis ke memori sehingga dapat digunakan kembali.

**Penghapusan simpul** **setelah simpul tertentu**

Operasi ini akan menghapus elemen setelah elemen tertentu. Skema penghapusan ini digambarkan sebagai berikut:

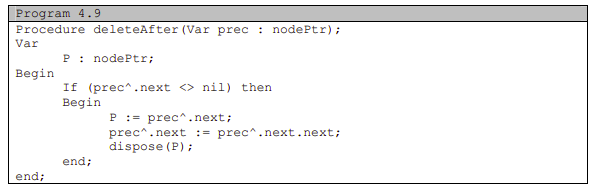


Langkah-langkah yang menjamin ketidakputusan rantai linked list adalah sebagai berikut:

1. Elemen tertentu yang dihapus dicat

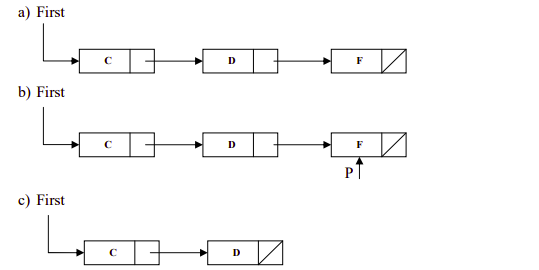
2. Sambungkan ke elemen berikutnya

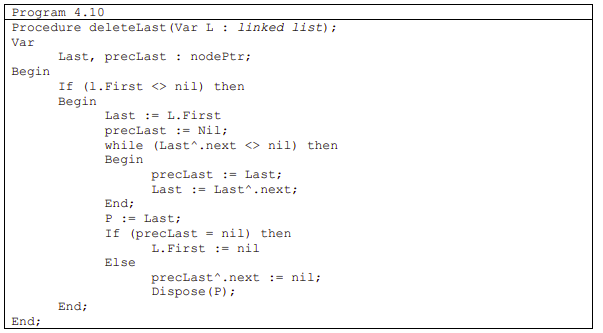
Kedua langkah tersebut mengisolasi elemen setelah elemen tertentu.



**Penghapusan simpul terakhir**

Operasi ini akan menghapus elemen pertama linked list. Sebelum memanggil prosedur ini harus dijamin linked list kosong. Skema penghapusan ini digambarkan sebagai berikut:





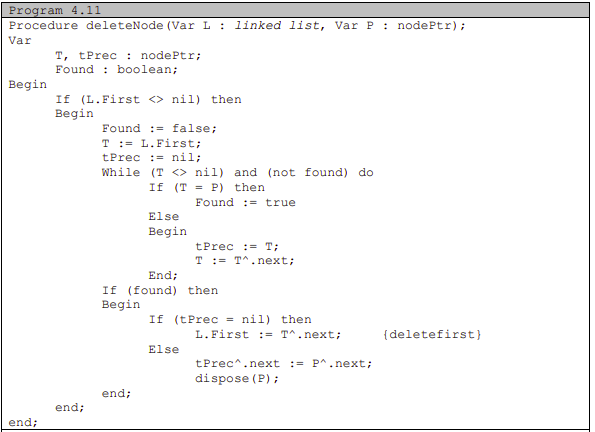
**Penghapusan simpul tertentu** Operasi ini akan menghapus elemen tertentu. Terdapat dua versi penghapusan ini, yaitu :

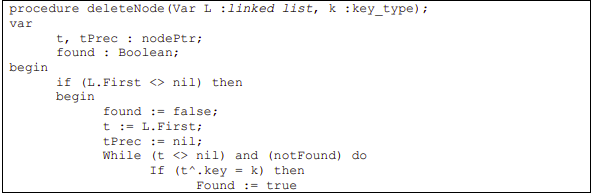
* Elemen tertentu didasarkan pada alamat elemen
* Elemen tertentu didasarkan pada key

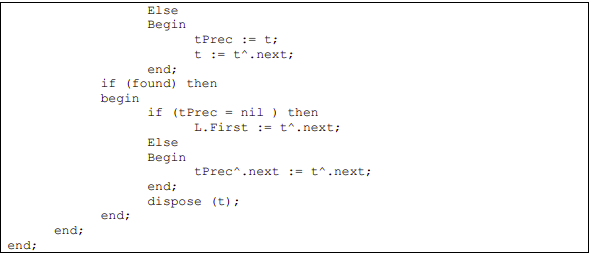
Untuk penghapusan elemen harus terlebih dulu diketahui elemen sebelum elemen tertentu tersebut dengan pencarian yang sekaligus mencatat elemen sebelumnya. Langkah-langkah yang menjamin ketidakputusan rantai linked list adalah sebagai berikut:

1. Menelusuri apakah terdapat elemen tertentu tersebut sekaligus mencatat elemen sebelumnya
2. Setelah ditemukan maka hapus elemen tertentu dengan

* Mengisolasi elemen tertentu
* Menghubungkan elemen sebelumnya dengan elemen setelah elemen tertentu · Dispose elemen tertentu



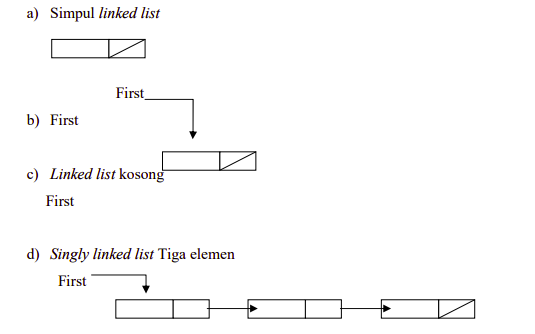


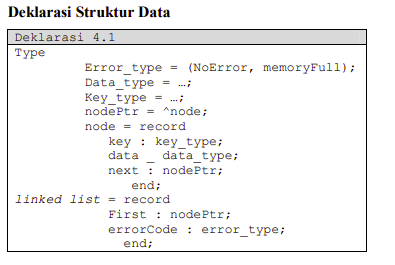


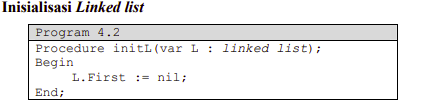
## **2.2 Jenis**

### 2.2.1 Single Linked List

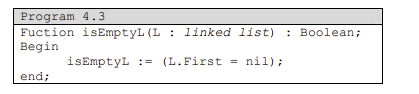
Single linked list adalahh sebuah linked list yang menggunakan variabel pointer saja untuk menyimpan banyak data dengan metote linked list, suatu daftar yang isinya saling berhubungan.







Procedure ini menghasilkan linked list kosong ditunjuk L.First, inisialisasi ini penting bila linked list juga memuat informasi-informasi lain. Untuk memeriksa linked list kosong adalah



**Representasi Single Linked List**

* Representasi Simpul

Typedef struct simpul Node;

Struct simpul {

Int data;

Node \*next;

};

* Deklarasi Global

Node \*head = null;

Node \*baru;

**Operasi pada Single Linked List**

* Operasi Sisip pada Single Linked List
  1. Penyisipan didepan

Penyisipan didepan pada single linked list adalah dengan cara menyisipkan data pada elemen awal list, sehingga pointer awal menunjuk list baru.

* 1. Penyisipan ditengah

Penyisipan ditengan pada single linked list adalah dengan cara menyisipkan data baru setelah elemen yang ditunjuk oleh variabel bantu pada list, kemudian pointer variabel baru menunjuk pointer variabel tertentu.

* 1. Penyisipan dibelakang

Penyisipan dibelakang pada single linked list adalah dengan cara menyisipkan data pada elemen akhir list, sehingga pointer akhir menunjuk list baru

* Operasi Hapus
  1. Menghapus simpul
  2. Hapus simpul awal
  3. Hapus simpul akhir
  4. Hapus simpul tertentu

### 2.2.2 Double Linked List

Double linked list adalah suatu linked list yang mempunyai 2 penunjuk yaitu penunjuk ke simpul sebelumnya dan simpul berikutnya.

Representasi Double Linked List

* Representasi Simpul

Typedef struct simpul Dnode;

Struct simpul {

Int data;

Dnode \*next;

Dnode \*prev;

};

* Deklarasi Global

Dnode \*head = null;

Dnode \*tail = null;

Dnode \*baru;

Operasi pada Double Linked List

* Operasi Sisip
  1. Penyisipan didepan

Operasi ini berguna untuk menambahkan satu simpul baru diposisi pertama. Langkah pertama untuk menambahkan data adalah pembuatan simpul baru dan mengisinya dengan data pada field infonya. Pointer yang menunjuk ke simpul tersebut dipanggil dengan nama baru.

* 1. Penyisipan ditengah

Operasi penyisipan data ditengah linked list adalah suatu operasi menambah data di posisi tertentu di dalam linked list. Karena double linked list memiliki dua pointer sambungan, maka penyisipan bisa dilakukan sebelum data tertentu atau sesudah data tertentu, berbeda dengan single linked list yang hanya memiliki satu pointer sambungan, yaitu sambungan next.

* 1. Penyisipan dibelakang

Operasi ini berguna untuk menambahkan elemen baru diposisi akhir. Langkah pertama untuk penambahan data adalah pembuatan elemen baru dan pengisian nilai infonya. Pointer yang menunjuk ke data tersebut dipanggil dengan nama baru.

* Operasi Hapus
  1. Hapus simpul awal

Penghapusan data diawal adalah proses penghapusan simpul pertama (yang ditunjuk oleh variabel pointer awal), sehingga variabel pointer awal akan berpindah ke simpul berikutnya. Ada tiga kondisi yang perlu diperhatikan, yaitu kondisi linked list masih kosong, kondisi linked list hanya memiliki satu data, dan linked list yang memiliki data ebih dari satu elemen.

* 1. Hapus simpul akhir

Penghapusan data diakhir adalah proses penghapusan pada simpul akhir (yang ditunjuk pointer akhir) dengan cara mengcopy pointer akhir dengan pointer hapus dan memindahkan pointer akhir ke tetangga kirinya dan datapun dapat dihapus.

* 1. Hapus simpul tengah

Penghapusan data ditengah merupakan proses penghapusan data dengan cara user memasukkan data/posisi data yang akan dihapus dan ditunjuk oleh pointer hapus. Untuk menghapus data tersebut dibutuhkan sebuah pointerbantu yang menunjuk data sebelum data yang akan dihapus. Dengan tujuan sebagai penyambung kembali data sebelum dan sesudah data yang akan dihapus.

### 2.2.3 Circular Linked List

Circular Linked list terbagi dua, yaitu

* Single Linked List Circular

Single linked list circular adalah single linked list yang pointer nextnya menunjuk pada dirinya sendiri. Jika single linked list tersebut terdiri dari beberapa node, maka pointer next pada node terakhir akan menunjuk ke node terdepannya.

Ilustrasi single linked list circular:

* 1. Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya, dan juga memiliki field yang berisi data.
  2. Pada akhir linked list node terakhir akn menunjuk ke node terdepan sehingga linked list tersebut berputar.
* Double Linked List Circular

Double linked list circular adalah linked list yang menggunakan pointer, dimana setiap node memiliki 3 field, yaitu:

* 1. Satu field pointer yang menunjuk pointer berikutnya “next”.
  2. Satu field pointer yang menunjuk pointer sebelumnya “prev”.
  3. Satu field pointer yang berisi data untuk node tersebut.

Double linked list circular pointer next dan prev nya menunjuk kedirinya sendiri secara circular. Ilustrasi double linked list circular:

* 1. 1. Setiap node pada linked list mempunya field yang berisi data dan pointer ke node berikutnya dan node sebelumnya.
  2. Untuk pembentuka node baru, mulanya pointer next dan prev akan menunjuk ke nilai null.
  3. Selanjutnya pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node selanjutnya pada list.

# BAB III

SOURCE CODE dan OUTPUT PROGRAM

#include

#include

struct node // deklarasi node sebagai struct yang isinya nama kelas dan npm

{

char nama[20];

char kls[6];

char npm[9];

node \*nxt; // Pointer untuk node selanjutnya

};

node \*mulai\_ptr = NULL;

node \*saat\_ini;

int pilihan = 0;

void tambah\_node\_di\_akhir()

{

node \*temp, \*temp2;

temp = new node;

cout << "Masukkan nama: ";cin >> temp->nama;

cout << "Masukkan Kelas : ";cin >> temp->kls;

cout << "Masukkan NPM : ";cin >> temp->npm;

temp->nxt = NULL;

if (mulai\_ptr == NULL)

{

mulai\_ptr = temp;

saat\_ini = mulai\_ptr;

}

else

{

temp2 = mulai\_ptr;

while (temp2->nxt != NULL)

{

temp2 = temp2->nxt;

} temp2->nxt = temp;

}

} void tampilkan\_list()

{

node \*temp;

temp = mulai\_ptr;

cout << endl;

if (temp == NULL)

cout << "List kosong!" << endl;

else

{

while (temp != NULL)

{

cout << "nama : " << temp->nama << " ";

cout << "Kelas : " << temp->kls << " ";

cout << "NPM : " << temp->npm;

if (temp == saat\_ini)

cout << " <-- saat\_ini node";

cout << endl;

temp = temp->nxt;

}

cout << "Akhir dari list!" << endl;

}

}

void delete\_mulai\_node()

{

node \*temp; temp = mulai\_ptr;

mulai\_ptr = mulai\_ptr->nxt;

delete temp;

}

void delete\_akhir\_node()

{

node \*temp1, \*temp2;

if (mulai\_ptr == NULL)

cout << "List kosong!" << endl;

else {

temp1 = mulai\_ptr;

if (temp1->nxt == NULL)

{

delete temp1;

mulai\_ptr = NULL;

}

else

{

while (temp1->nxt != NULL)

{

temp2 = temp1;

temp1 = temp1->nxt;

}

delete temp1;

temp2->nxt = NULL;

}

}

} void move\_saat\_ini\_on ()

{

if (saat\_ini->nxt == NULL)

cout << "Kamu berada pada akhir list." << endl;

else saat\_ini = saat\_ini->nxt;

}

void move\_saat\_ini\_back ()

{

if (saat\_ini == mulai\_ptr)

cout << "Kamu berada pada awal list" << endl;

else

{

node \*sebelum;

sebelum = mulai\_ptr;

while (sebelum->nxt != saat\_ini)

{

sebelum = sebelum->nxt;

}

saat\_ini = sebelum;

}

}

void main()

{

mulai\_ptr = NULL;

do

{

clrscr();

tampilkan\_list();

cout << endl;

cout << "-----------------------------------------------------------\n";

cout << "Masukkan pilihan Anda : " << endl;

cout << "-----------------------------------------------------------\n";

cout << "0. Keluar dari program." << endl;

cout << "1. Tambah node di akhir list." << endl;

cout << "2. Hapus awal node dalam list." << endl;

cout << "3. Hapus akhir node dalam list." << endl;

cout << "4. Pindah pointer saat\_ini ke depan satu node." << endl;

cout << "5. Pindah pointer saat\_ini ke belakang satu node." << endl;

cout << endl << " >> ";

cin >> pilihan;

switch (pilihan)

{

case 1 :

tambah\_node\_di\_akhir();

break;

case 2 :

delete\_mulai\_node();

break;

case 3 :

delete\_akhir\_node();

break;

case 4 :

move\_saat\_ini\_on();

break;

case 5 : move\_saat\_ini\_back();

}

}

while (pilihan != 0);

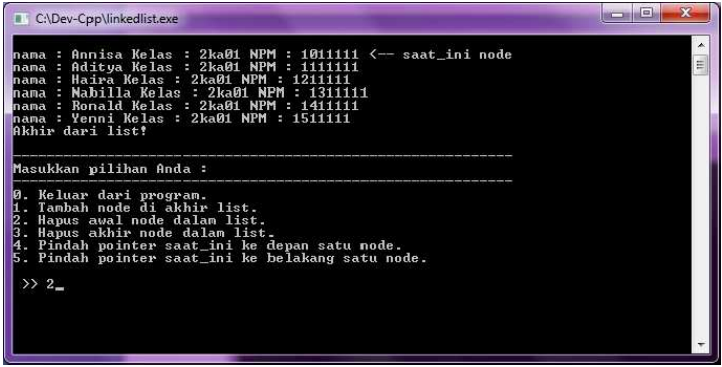
}

OUTPUT

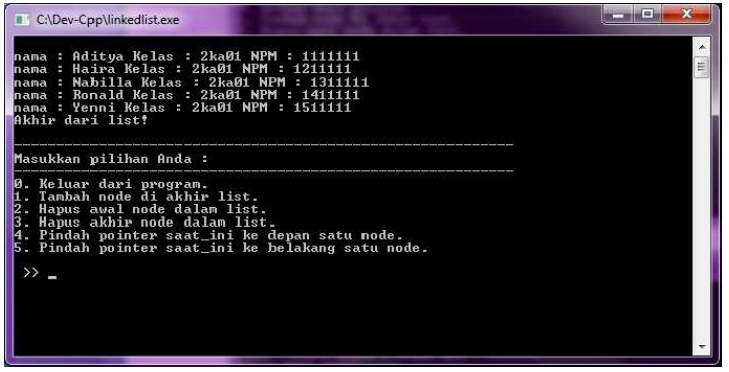
1. Berikut adalah output dari contoh program linked list , bila di input pilihan no. 1 maka akan tampil menu untuk memasukkan nama , kelas, dan npm.



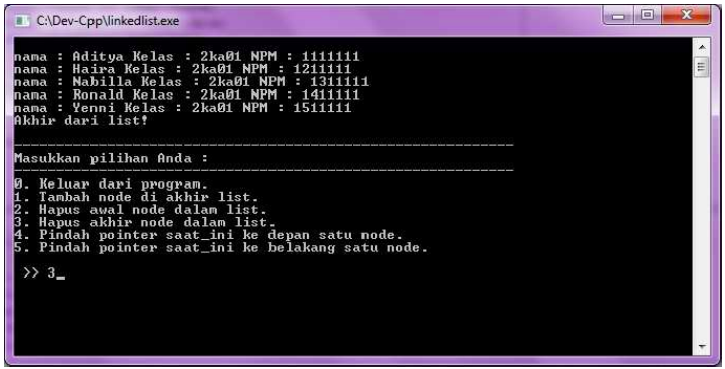
1. Setelah selesai diinput , selanjutnya input pilihan no.2 yaitu pilihan untuk menghapus awal node dalam list .



1. Tampilan berikutnya seperti dibawah ini .



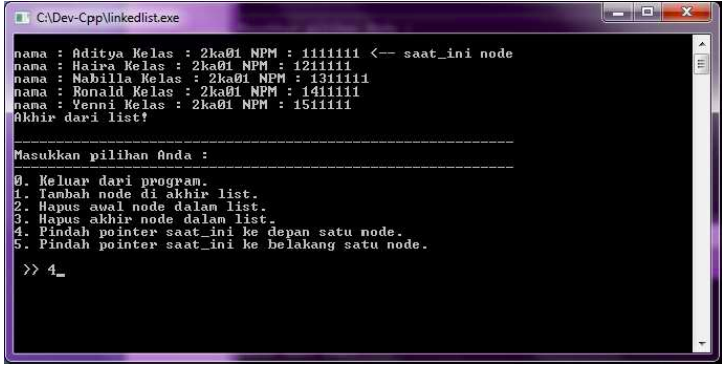
1. Input no.3 yaitu menghapus node paling terakhir ,



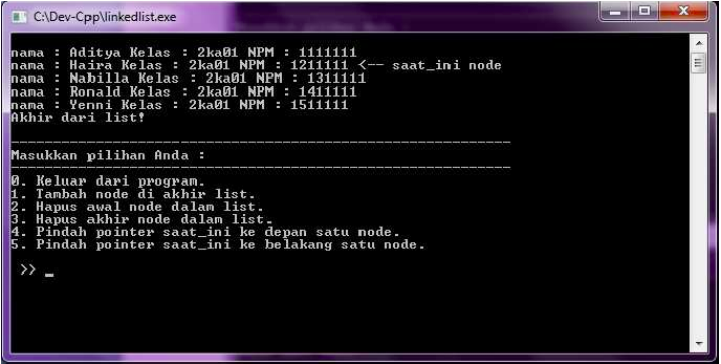
1. Tampilannya seperti berikut



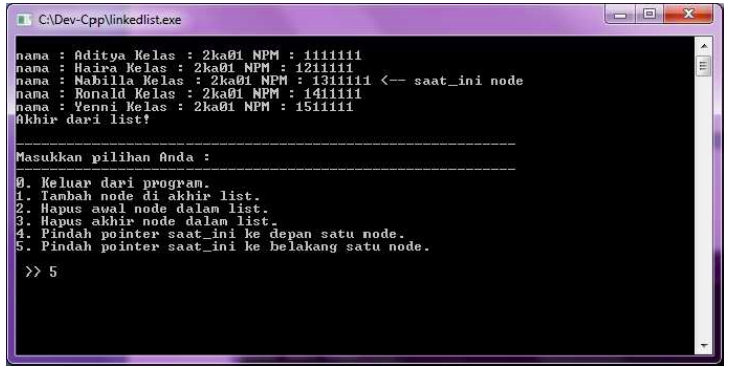
1. Perhatikan pointer yang berada di atas , saat di input no.4 , pointer akan berada didepan satu node.



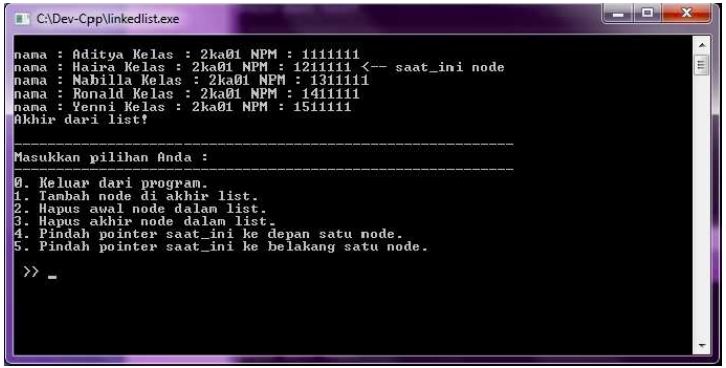
1. Seperti tampilan di bawah ini .



1. Perhatikan pointer , setelah di input no.5 , pointer akan berada di belakang sebanyak satu node



1. Tampilan seperti dibawah ini



1. Setelah selesai melakukan proses linked list , maka bisa menginput pilhan 0 untuk keluar dari program .

# BAB V

PENUTUP

**kesimpulan**

Setelah melakukan penelitian ini, dapat diambil kesimpulan sebagai berikut :

1. 1. Struktur data telah berhasil dibuat sebagai tempat penyimpanan nodes yang terdeteksi pada citra labirin yang digunakan untuk input algoritma Dijkstra dengan memanfaatkan struktur data Double Linked List I.
2. 2. Aplikasi pada komputer sebagai pengolah citra labirin dan pencarian nodes pada citra labirin telah berhasil dibuat. Aplikasi yang dibuat dapat mendeteksi seluruh simpangan atau jalan buntu sebagai node jika lebar jalur labirin merata dan akan mengalami kegagalan saat proses pencarian nodes jika terdapat lebar jalur labirin yang tidak merata.
3. 3. Pengendali Micromouse Robot telah berhasil dibuat tetapi pada pergerakan Micromouse Robot tidak sepenuhnya mampu melalui nodes rute terpendek hasil pengolahan menggunakan algoritma Dijkstra dari titik start ke finish. Pergerakan Micromouse Robot sering terjadi kesalahan pada pendeteksian tikungan atau simpangan disebabkan ketidakstabilan sensor sehingga data yang didapat dari sensor tidak sesuai dengan yang diharapkan

**Saran**

Sebagai pengembangan dari penelitian yang telah dilakukan, penulis memberikan saran sebagai berikut:

1. Jika pada labirin terdapat lebih dari 254 node, agar paket data nodes dapat diterima oleh Micromouse Robot maka sebaiknya menggunakan mikrokontroler dengan kapasitas EEPROM yang lebih besar, dengan menambahkan EEPROM eksternal, ataupun dengan menghubungkan dengan media penyimpanan data yang lain, seperti memory card.
2. Pencarian node tidak dapat dilakukan jika dinding tepian labirin pada citra tidak sejajar dengan camera window. Perlu dikembangkan metode untuk memperbaiki posisi labirin pada citra jika terdapat kemiringan yang signifikan.
3. Pencarian node pada citra labirin tidak dapat dilakukan jika terdapat lebar lintasan yang tidak merata. Perlu dikembangkan suatu metode untuk mendeteksi node pada citra lintasan meskipun lebarnya tidak merata.
4. Penyelesaian masalah rute terpendek dapat dicoba dengan menggunakan metode yang lain.
5. Pengembangan selanjutnya dapat dicoba dengan menggunakan lintasan labirin dengan ukuran yang bervariasi.

Sensor dinding yang digunakan sebaiknya menggunakan sensor yang tidak terpengaruh cahaya dan warna dinding labirin.