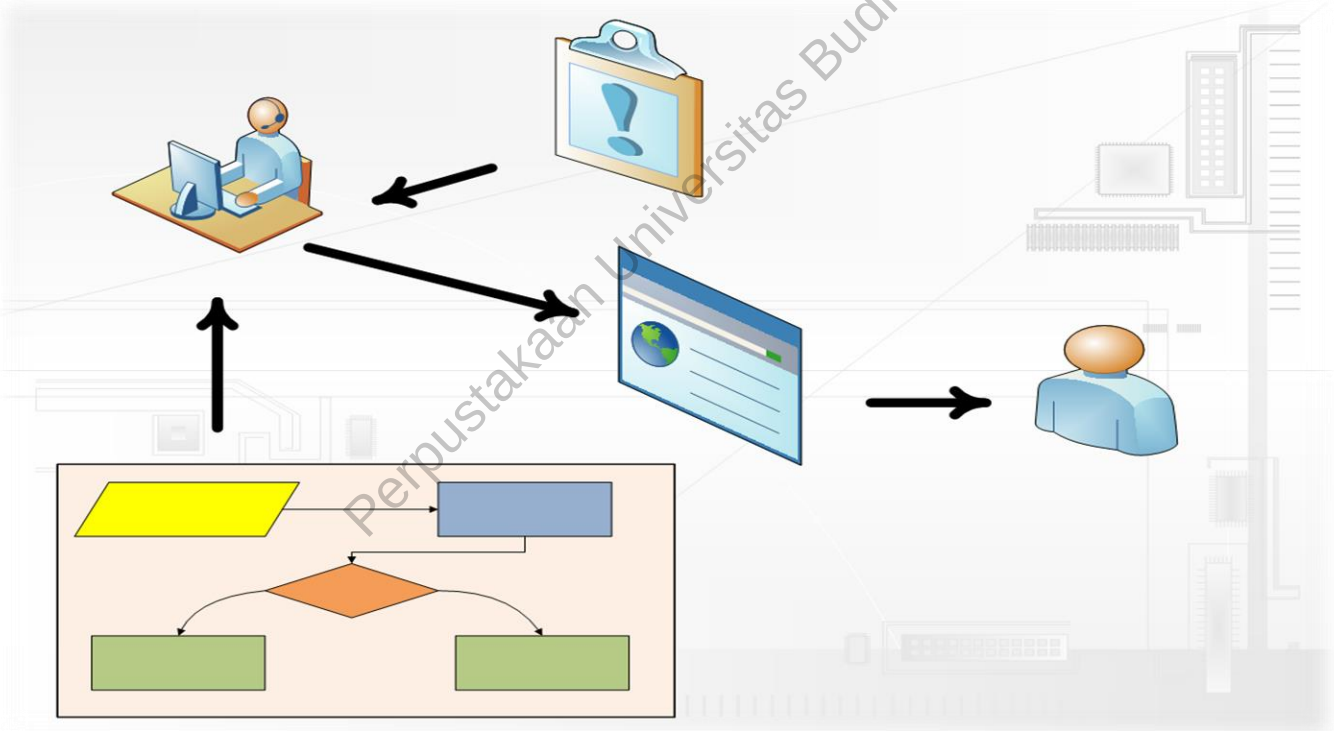
STRUKTUR DATA

DASAR DASAR PEMROGRAMAN



Hari Soetanto

Sanksi Pelanggaran Pasal 113
Undang-Undang Republik Indonesia Nomor 28 Tahun 2014 Tentang Hak Cipta

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).

2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).

3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah).

4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah).

STRUKTUR DATA DASAR-DASAR PEMROGRAMAN

Hari Soetanto

PENERBIT
UNIVERSITAS BUDI LUHUR

Struktur data dasar-dasar pemrograman

Penulis : Hari Soetanto

Diterbitkan oleh :
Penerbit Universitas Budi Luhur
Jalan Raya Ciledug Raya, Petukangan Utara, Jakarta Selatan 12260
DKI Jakarta, Indonesia
Telp. (021) 5853753
Fax. (021) 7371164
Email: Penerbit@budiluhur.ac.id

Cetakan 1, Jakarta, Januari 2022
ISBN: 978-623-92135-3-4

Editor : Pustaka Budi Luhur
Desain Cover : Hari Soetanto

Kata Pengantar

Puji syukur dipanjatkan kepada Tuhan Yang Maha Esa atas berkah, Rahmat dan bimbinganNya dalam penyusunan dan penyelesaian buku struktur data dasar dasar pemrograman ini.

Buku ini disusun dengan tujuan

- Pemahaman pengetahuan tentang teori dasar struktur data dan penanganan data.
- Pembuatan algoritme dan penggunaan struktur data dalam pemrograman.
- Implementasi ke dalam bahasa pemrograman C.
- Dasar-dasar dari algoritme untuk materi teknologi komputer, seperti artificial intelligence, Expert system, Data minning, algoritme dan pemrograman.

Sebagai manusia yang tidak luput dari kekurangan dan kesalahan, kami sadar bahwa buku ini masih jauh dari sempurna, oleh karena itu masukan -masukan diharapkan agar dapat melengkapi buku ini

Terimakasih dan Salam
Jakarta, Juli 2021

Penulis

Sipnosis

Buku ini cocok digunakan untuk mahasiswa, pelajar atau pemula yang bermaksud untuk mempelajari dasar-dasar struktur data dan implementasinya. Jenis jenis tipe data serta operasinya, array, linked list, stack, queue pengurutan (sort) dan pencarian (search). Contoh implementasi menggunakan bahasa pemrograman Bahasa C.

Dasar-dasar dari algoritme untuk materi teknologi komputer, seperti artificial intelligence, Expert system, Data minning, algoritme dan pemrograman.

Sebagai manusia yang tidak luput dari kekurangan dan kesalahan, kami sadar bahwa buku ini masih jauh dari sempurna, oleh karena itu masukan-masukan diharapkan agar dapat melengkapi buku ini

Daftar Isi

Definisi Data

Data:

Adalah fakta atau kenyataan yang tercatat mengenai suatu obyek

Pengertian data ini menyiratkan suatu nilai yang bisa dinyatakan dalam bentuk konstanta atau variabel.

- **Konstanta** menyatakan nilai yang tetap.
- **Variabel** digunakan dalam program untuk menyatakan nilai yang dapat diubah-ubah selama eksekusi berlangsung

1. Ada 4 istilah tentang Data

Tipe data:

macam/isi data di dalam suatu variabel dalam bahasa program

Obyek data:

set dari elemen, misal X set bilangan integer.

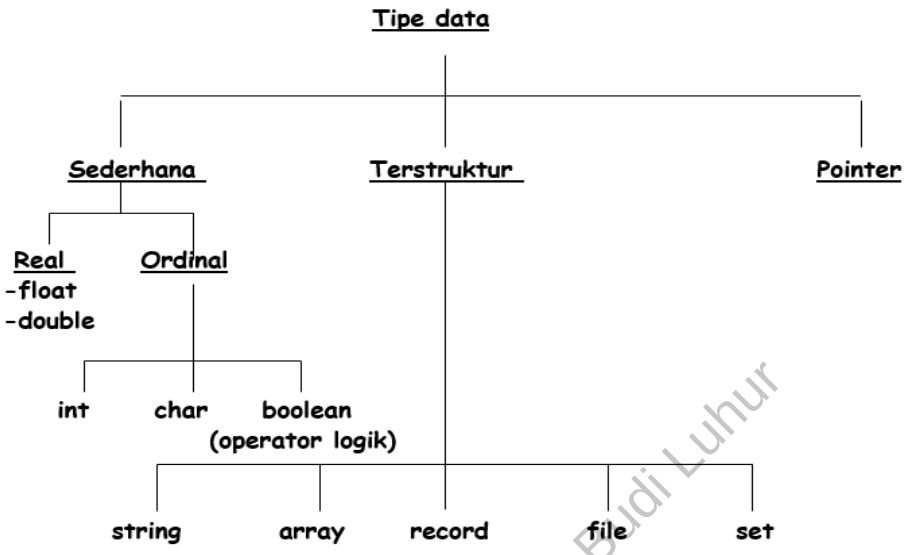
Representasi data:

Suatu mapping dari struktur data kesuatu set dari struktur data, misal boolean direpresentasikan dalam 0 dan 1.

Struktur data:

Struktur adalah koleksi dari variabel yang dinyatakan dengan sebuah nama, dengan sifat setiap variabel dapat memiliki tipe yang berlainan Struktur data biasa dipakai untuk mengelompokkan beberapa informasi yang berkaitan menjadi sebuah kesatuan.

2. Hirarkhi dari tipe data:



3. Tipe Data Sederhana

Hanya dimungkinkan untuk menyimpan sebuah nilai data dalam sebuah variabel .

Ada 5 macam :

- Bilangan bulat (integer)
- Bilangan real presisi-tunggal (float)
- Bilangan real presisi-ganda (double)
- Karakter
- Tak-bertipe
- Boolean (operator logika)

Tipe	Total bit	Jangkauan	Keterangan
Char	8	-128 s/d 127	karakter
Int	16	-32768 s/d 32767	bilangan integer
Float	32	3.4E-38 s/d 3.4E+38	bil.real - Presisi tunggal
Double	64	1.7E-308 s/d 1.7E+308	bil.real - Presisi ganda
Void	6		tak bertipe

4. Tipe Data Boolean

mempunyai 2 nilai: TRUE dan FALSE

Operator	Maksud
&&	Dan (AND)
||	atau (OR)
!	Tidak (NOT)

Operator logika biasa dipakai untuk menghubungkan ungkapan relasi. Baik operand1 maupun operand2 dapat berupa ungkapan relasi ataupun ungkapan logika. Hasil ungkapan bernilai benar (true) atau salah (false)

Operand1	Operand2	Hasil ||	Hasil &&
salah	salah	salah	salah
salah	benar	benar	salah
benar	salah	benar	salah
benar	benar	benar	benar

contoh:

```
if  (pilihan == 'Y') ||  (pilihan == 'y')
        {.........}
```

5. Tipe Data Terstruktur

adalah tipe dimana suatu variabel bisa menyimpan lebih dari sebuah nilai data. Masing-masing nilai data disebut komponen.

Ada 5 macam:

a. Tipe String

data yang berisi sederetan karakter dimana banyaknya karakter bisa berubah-ubah sesuai kebutuhan.

Bentuk umum:

<u>char</u> *nama_variabel*[ukuran];

contoh: char nama[30];

b. larik (array)

dimana variabel larik hanya bisa menyimpan 1 tipe data saja.

Bentuk umum :

tipe data *nama_variabel* [ukuran];

contoh: float A[10];
int X[5][5], Y[10];

c. record

terdiri dari beberapa variabel yang terstruktur dan masing-masing variabel bisa mempunyai tipe yang berbeda.

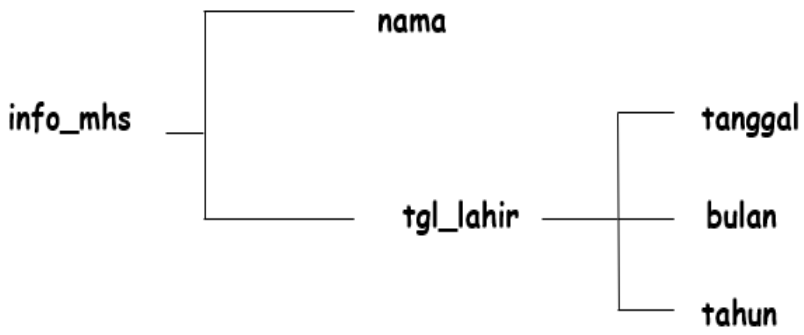
Bentuk umum :

```
struct nama_tipe_struktur
{ tipe field1;
  tipe field2;
      ............;
  tipe fieldn;
} var_struk1, var_struk2,....,var_strukn;
```

contoh struktur data dari info_mhs :

```
struct data_tanggal
{ int tanggal;
  int bulan;
  int tahun;
};

struct data_mhs
{     char nama[25];
      struct data_tanggal tgl_lahir;
} info_mhs;
```



d. set (himpunan)

1. **union**

   memungkinkan suatu lokasi memori ditempati oleh dua atau lebih variabel yang tipenya bisa berlainan.

   Bentuk umum:

```
Union  nama_union
     {
       tipe  field1;
       ............ ;
       tipe  fieldn;
      } var_union1,  var_union2;
```

   contoh:

```
union
{
  unsigned  int dat_int;
  unsigned  char  dat_char[2];
} bil_x;
```

2. **enumerasi**

   merupakan himpunan dari konstanta integer yang diberi nama.

Bentuk umum :

```
Enum  nama_enum
{  konstanta_1,  konstanta_2, ….,
   konstanta_n} var_1, var_2,…,  var_n;
```

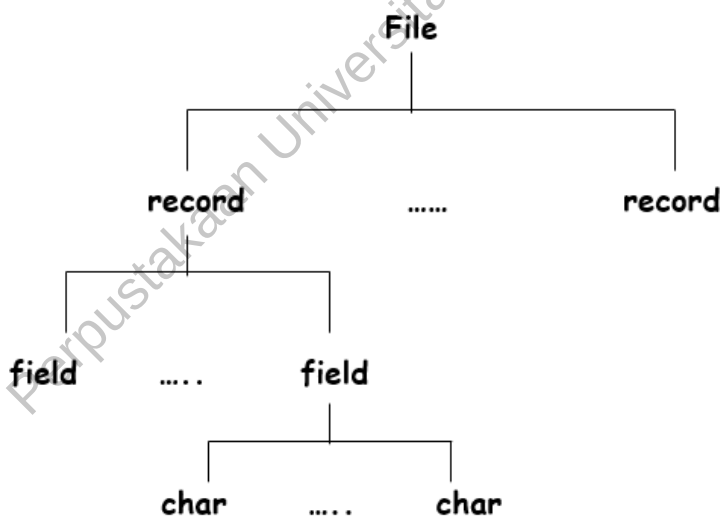
contoh :

```
enum  manusia  { pria,  wanita};
enum  manusia  jns_kelamin;
```

bila jns kelamin diisi pria maka nilai jns_kelamin = 0 dan sebaliknya bila diisi wanita nilai = 1;

e. file

Merupakan organisasi dari sejumlah record sejenis. Masing-masing record dapat terdiri dari satu atau beberapa field dan setiap field terdiri dari satu atau beberapa karakter.



6. Tipe Data Pointer

Variabel pointer berisi alamat dari suatu obyek lain (yaitu obyek yang ditunjuk oleh pointer tersebut).

Bentuk umum:

tipe **nama pointer;*

Contoh:

```
int   *pa;
pa = &x;
```

pointer pa menunjuk alamat x.

Operasi Data

1. Tipe data sederhana

a. Tipe data integer

Menempati memori sepanjang 2 byte, dan merupakan bilangan bertanda.

```
#include <stdio.h>

main()  {
  int  int1, int2;

  clrscr();
  printf("\n\nmasukkan data :\n\r int1 = ");
  scanf(" %d" ,&int1);
  printf(" \n\r int2 = ");
  scanf(" %d" , &int2);

  printf("\n\r int1 = %d  heksa= %x alamat di = %x", int1,int1,&int1);
  printf("\n\r int2 = %d  heksa= %x alamat di = %x", int2,int2,&int2);
  getch();
  return(0);
}
```

Hasil:

```
masukkan   data  :
int1  =  100
int2 =  -100
int1  =  100    heksa  =  64      alamat  di  ffe
int2  =  -100   heksa  =  ff9c    alamat  di  ffc
```

b. Tipe Data Real (float)

Perbedaan antara tipe float dan double selain jangkauan nilai dimana double lebih besar dari float, juga ketelitiannya. Tipe float hanya mempunyai ketelitian (mantisa) sampai 7 digit, sedangkan tipe double memiliki ketelitian sekitar 16 digit.

Float menempati 4 byte, sedang double menempati 8 byte.

```
#include <stdio.h >

main()   {
         float    x;
         double y;
         x  =  56.0123456789123456789;
         y  =  56.0123456789123456789;

         printf("X = %20.16f\n", x);
         printf("Y = %20.16f\n", y);
         getch();
         return 0;
       }
```

Terlihat outputnya sebagai berikut :

X = 56.0123443603515625
Y = 56.0123456789123466

Untuk melihat pemakaian memori baik float maupun double bisa dicoba dari program dibawah ini.

```
#include <stdio.h>

main() {
     float fl1, fl2; double db1, db2;
     clrscr();
     printf("\n\nmasukkan data :\n\r fl1 = ");
     scanf(" %f" ,&fl1);

     printf(" \n\r fl2 = ");
     scanf(" %f" , &fl2);

     printf(" \n\r db1 = ");
```

```
    scanf(" %lf" , &db1);
    printf(" \n\r db2 = ");
    scanf(" %lf" , &db2);

    printf("\n\rharga fl1 = %f alamat di = %x ", fl1, &fl1);
    printf("\n\rharga fl2 = %f alamat di = %x ", fl2, &fl2);
    printf("\n\rharga db1 = %20.16lf alamat di = %x ", db1, &db1);
    printf("\n\rharga db2 = %20.16lf alamat di = %x ", db2, &db2);
    getch();
    return(0);
}
```

Latihan: Bagaimana bentuk outputnya.

2. Tipe Data karakter

Tipe data karakter ini menggunakan kode ASCII. Untuk tipe data sederhana menempati 1 byte sedang tipe data string, satu nama variabel char dapat berisi lebih dari satu data.

```
#include <stdio.h>

main(){
  char char1, char2, char3[8];
  clrscr();
  char2 = 'k';
  printf("\n\nmasukkan data :\n\r char1 = ");
  scanf(" %c" , &char1);
  printf(" \n\r char3[9] = ");
  scanf(" %s" , &char3);

 printf("\n\rchar1 = %c ascii = %x alamat di = %x",char1,char1,&char1);
```

```
  printf("\n\rchar2 = %c ascii = %x alamat di = %x",char2,char2,&char2);
  printf("\n\rchar3 = %s ascii = %x alamat di = %x",char3,char3[1],&char3);

  getch();
 return(0);
}
```

Latihan:
tunjukkan nilai hexa dan alamat dari char1, char2 dan char3

Algoritme

1. Definisi Algoritme

adalah himpunan langkah-langkah instruksi untuk melaksanakan suatu pekerjaan tertentu, dengan beberapa kriteria:

a. ada input
b. ada output
c. jelas dan tidak meragukan (definiteness)
d. ada terminasi (finiteness)
e. efektif dan dapat dilaksanakan

Ada sedikit perbedaan antara algoritme dan program.

Program tidak harus memenuhi kriteria 4, contohnya Sistem Operasi (OS) tidak memenuhi kriteria tersebut karena harus selalu menunggu *task*.

2. Algoritme dan Metode Pemrograman

Penekanannya pada bagaimana memecahkan suatu masalah dengan algoritme yang tepat.

Dasar-dasar algoritme :
- statement elementer
- statement control

Statement elementer :
- assignment
- comparison
- arithmatic statement
- operator Boolean
- instruksi I/O

Statement Control :
- Alternatif
- Pengulangan
- Percabangan

3. Statement Elementer (dasar)

a. **assignment**

Untuk memberikan nilai ke variabel yang telah dideklarasikan, bentuk pernyataan yang digunakan :

Variabel = nilai;

Contoh: total = 0;

b. **Comparison**

Untuk keperluan pengambilan keputusan diperlukan operator relasi sebagai berikut :

Operator	Arti
>	Lebih dari
>=	Lebih dari atau sama dengan
<	Kurang dari
<=	Kurang dari atau sama dengan
==	Sama dengan
!=	Tidak sama dengan

c. **Arithmetic statement**

Operator untuk operasi aritmatika adalah :

^ pangkat

* , / perkalian, pembagian

+ , - penjumlahan, pengurangan

ada operator aritmatika yang pemakaiannya adalah khusus, yaitu:
% sisa pembagian

contoh: (7 % 2 → hasilnya 1)

d. **Operator Boolean**

Operator Boolean atau operator logika biasa dipakai untuk menghubungkan ungkapan relasi yang hasilnya adalah *True* dan *False*, yaitu:

&&	AND	(dan)
||	OR	(atau)
!	NOT	(tidak)

e. **Operasi input/output**

Untuk memasukkan data (input data) ke komputer dapat melalui instruksi (dalam bahasa C) sebagai berikut:

scanf(),
getch(),
getche()

Sedang untuk mengeluarkan (output) data digunakan :

printf(),
puts(),
putchar().

3. Statement Control

a. Alternatif

Terdiri dari pernyataan :
- If
- if - else
- switch

Bentuk umum :

➢ if (kondisi)
{ pernyataan }

➢ if (kondisi)
{ pernyataan-true }
else
{ pernyataan-false }

➢ switch (ekspresi)
{
case-1: pernyataan-1
break;
……

```
        case-n: pernyataan-
                break;
    }
```

b. Pengulangan

Pernyataan berulang terdiri dari:

- do-while
- while
- for

Bentuk umum :

- ```
  do
    { pernyataan }
  while ( kondisi )
  ```
- ```
  while ( kondisi )
    { pernyataan }
  ```
- ```
  for ( ungkapan1 ; ungkapan2 ; ungkapan3)
    { pernyataan }
  ```

c. Percabangan

Memerlukan label sebagai identitas cabang.

Bentuk umum :

```
label:
{
  pernyataan
}
goto  Label
```
```

Struktur Data Linier

Struktur data linier adalah struktur data yang menggambarkan hubungan tentang elemen-elemen yang berdekatan.

Terdiri dari :

- **ARRAY** : a. dimensi satu (vektor matriks)
  b. dimensi dua (matriks)
  c. multi dimensi

Aplikasi penggunaan array diantaranya adalah :

1. stack (tumpukan)
2. queue (antrian)
3. deque (antrian dengan dua pintu)

1. Linked list (list berkait)
   a. linear singly linked list
   b. linear doubly linked list
   c. circular singly linked list
   d. circular doubly linked list

Aplikasi linked list pada struktur data linier diantaranya:

1. linked stack
2. linked queue

Sedang multi linked list banyak digunakan pada struktur data non-linier yaitu untuk representasi tree maupun graph.

2. Pengelolaan Memori

Dapat secara STATIS atau DINAMIS.

- **Secara STATIS :**

  menempati lokasi memori yang tetap (fixed size), tidak dapat dikembangkan atau diciutkan.

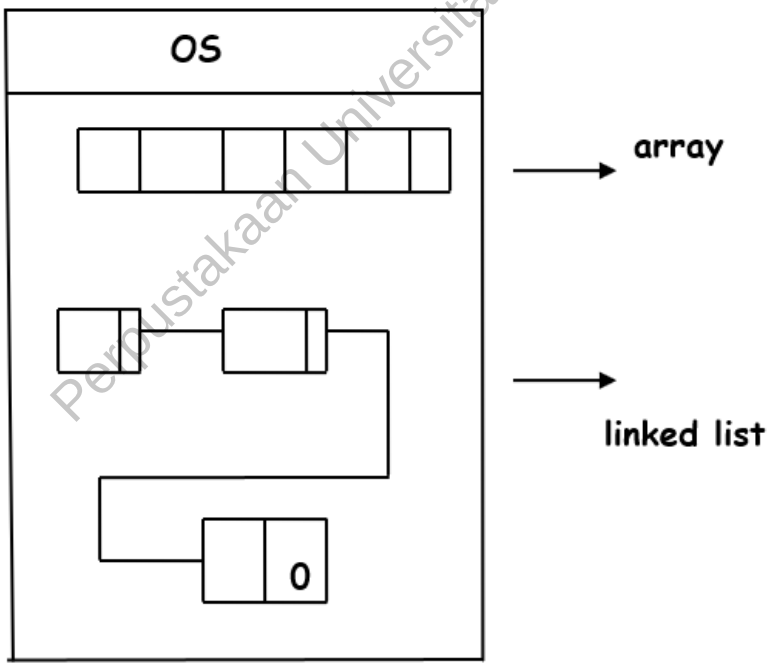
  Misal : array

  alamat memori menjadi kunci array

- **Secara DINAMIS :**

  menempati lokasi memori dimana dapat dikembangkan atau diciutkan sesuai dengan kebutuhan.

Pengelolaan alamat dinamis (dynamic address) ditunjukkan oleh pointer.



Array (Larik)

Array (larik) adalah tipe terstruktur yang terdiri dari sejumlah komponen - komponen dengan tipe yang sama.
Banyaknya komponen dalam satu larik adalah tetap dan lokasi dalam suatu larik ditunjukkan oleh suatu INDEKS.
Yang penting dalam array adalah pengalamatan memori dan digunakan pengalamatan secara statik.

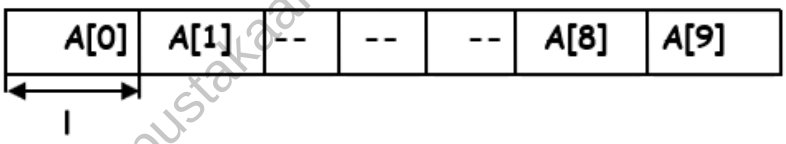
1. Karakteristik pemakaian array

- jumlah elemen array terbatas
- semua elemen array bisa diakses secara acak
- panjang elemen sama.

Contoh: **int A[10];**
float B[5];

Matriks A akan dialokasikan 10 tempat dalam memori dengan panjang yang sama

Array dimensi satu dengan 10 lokasi :



Lokasi untuk elemen ke-1 dari array A :

@ A[i] = @ A[0] + l (i - b)

dimana: @ A = alamat
l = # lokasi memori untuk setiap elemen
b = batas bawah

contoh: @A[5] = @A[0] + l (5-0)

bila @A[0] = 1000H dan l= 1 byte,
maka :
@A[5] = 1000H + 1(5-0)
= 1005 H

Aplikasi dalam bahasa pemrograman C:

```
#include <stdio.h >

main() {
   int i ;
   static int B[5]  = { 130, 135, 70, 180, 90 };
   static float C[5]= { 4, 3, 2, 1.5, 7};
   clrscr();
   printf("\n");
   printf("\n\r ");
   for (i = 0; i< 5; i++)  {
      printf("\nB[%i] = %i , heksa = %x ada di %x\n", i, B[i],B[i],&B[i]);
  }
  printf("\n\r");
  for (i = 0; i< 5; i++)   {
         printf("C[ %i ] = %f , %x\n",i, C[i],&C[i]);
   }
   getch();
   return(0);
 }
```

Akan terlihat output dari program diatas sebagai berikut :

```
B[0] = 130,    ada di 90
B[1] = 135,    ada di 92
B[2] = 70,     ada di 94
B[3] = 180,    ada di 96
B[4] = 90,     ada di 98

C[0] =  4.000000 ,    ada di 9a
C[1] =  3.000000 ,    ada di 9e
C[2] =  2.000000 ,    ada di a2
C[3] =  1.500000 ,    ada di a6
C[4] =  7.000000 ,    ada di aa
```

2. **Array dimensi 2**

representasi matriks di memori dalam bentuk array satu dimensi, dapat berupa:

- ukuran baris per baris (Row major order)
- ukuran kolom per kolom (Column major order)

Matriks dalam pembahasan selanjutnya dengan pernyataan

M[baris] [kolom]

COLUMN - MAJOR ORDER (representasi kolom per kolom)

@ M[i][j] = @ M[0][0] + { (i- 0) + (j - 0)k } l

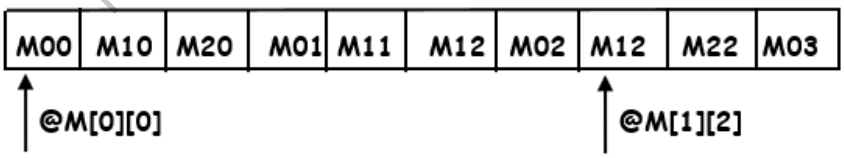
dimana : i = baris
j = kolom
l = panjang elemen
k = banyaknya elemen/kolom

Untuk matriks dengan ukuran M[k][n]

M[0][0]	M[1][0]	.....	M[k][0]	M[0][1]	M[1][1]	...	M[k][1]

Contoh: Tunjukkan lokasi M[1][2] dari matriks M[3][4] jika l=1 dengan menggunakan column-major order

Jawab: **@M[1][2] = @M[0][0] + {(1-0)+ (2-0) 3 } 1**
= @M[0][0] + 7



ROW - MAJOR ORDER (representasi baris per baris)

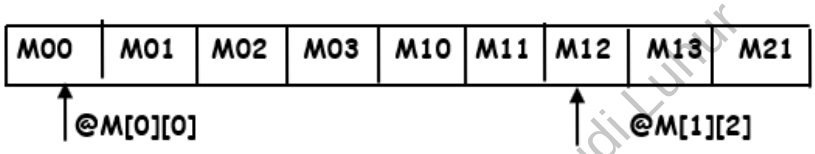
@M[i][j] = @M[0][0] + {(i-0) n + (j-0)} l

dimana besaran-besaran lain sama dengan kolom per kolom, hanya n = banyaknya elemen per baris.

M[0][0]	M[0][1]	….	M[0][n-1]	M[1][1]	M[1][2]	….	M[1][n-1]

contoh : tunjukkan lokasi M[1][2] dari matriks M[3][4] dengan menggunakan row-major order dengan l = 1 byte.

Jawab : @M[1][2] = @M[0][0] + { (2-0) + (1-0) 4} 1
= @M[0][0] + 6



Aplikasi Program dalam Bahasa C :

```
#include <stdio.h>
 main() {
   int i, j;
   static float M[4][3] = { 0,1,2,  10,11,12,  20,21,22,  30,31,32 };

   for (i = 0; i<4; i++)    {
     for (j=0; j<3; j++)    {
        printf("M[%i][%i] = %f di alamat %X\n", i, j, M[i][j], &M[i][j]);
      }
     printf("\n");
    }

    getch();
    printf("M[1][2] = %f ada di %X\n", M[1][2], &M[1][2]);
    getch();
    return(0);
  }
```

Latihan: Ubah ukuran menjadi M[5][4], isi datanya. Tampilkan lokasi M[4][3].

3. Representasi multidimensional array

Memori bisa dipandang sebagai satu dimensional array dengan alamat dari 1 .sampai dengan. m

Untuk menggambarkan n dimensi dalam satu dimensional array maka digunakan pernyataan sebagai berikut :

Pernyataan array:

A (l1 : u1 , l2 : u2 , ……. , ln : un)

dimana : l1 = harga awal dari dimensi-1
u1 = harga akhir dari dimensi-1
l2 = harga awal dari dimensi-2
u2 = harga akhir dari dimensi-2

Jumlah elemen array: $\prod_{i=1}^{n} (u_i - l_i + 1)$

Alamat dari A[i1, i2, i3, ….. , in] adalah :

$$A[i_1, i_2, \ldots, i_n] = \alpha + \{ \sum_{j=1}^{n} (i_j - l_j)\, a_j \} L$$

$$\text{dengan} \quad aj \quad \begin{cases} \prod_{k=j+1}^{n} uk & ; \quad 1 <= j < n \\ an = 1 \end{cases}$$

dimana : α = elemen pertama dari array
= A[l1 , l2 , ... , ln]
uk = jumlah elemen dalam dimensi i ke k
L = panjang elemen dalam satuan byte

Contoh: multidimensional array A(4:5 , 2:4 , 1:2 , 3:4) alamat elemen pertama α atau A[4,2,1,3] ada di 1800 H dan L = 1 byte.
Cari: a. jumlah elemen array
b. alamat elemen A[5,2,1,4]

Dengan menggunakan row-major order, elemen disimpan dalam urutan di memori sebagai berikut :

A[4,2,1,3] , A[4,2,1,4] , A[4,2,2,3] , A[4,2,2,4]
A[4,3,1,3] , A[4,3,1,4] , A[4,3,2,3] , A[4,3,2,4]
A[4,4,1,3] , A[4,4,1,4] , A[4,4,2,3] , A[4,4,2,4]
A[5,2,1,3] , A[5,2,1,4] , A[5,2,2,3] , A[5,2,2,4]
A[5,3,1,3] , A[5,3,1,4] , A[5,3,2,3] , A[5,3,2,4]
A[5,4,1,3] , A[5,4,1,4] , A[5,4,2,3] , A[5,4,2,4]

Jawaban:

a. jumlah elemen = (5-4+1) . (4-2+1) . (2-1+1) . (4-3+1)
= 2 . 3 . 2 . 2
= 24

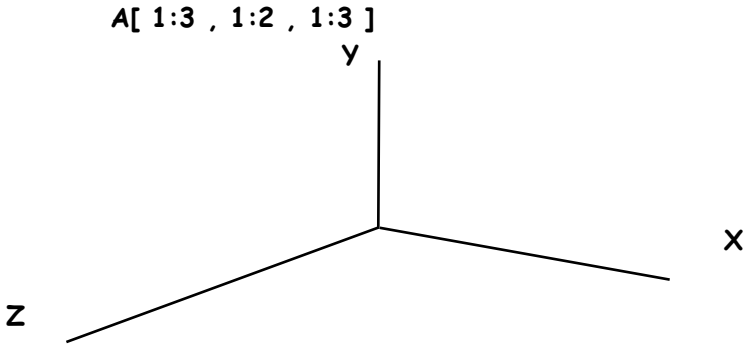
b. alamat A[5,2,1,4] = A[4,2,1,3] + { (5-4) . 3 . 2 . 2 +
(2-2) . 2 . 2 +
(1-1) . 2 +
(4-3) . 1 } 1
= 1800 H + { 12 + 0 + 0 + 1 } 1
= 1800 H + 13
= 180DH

Aplikasi dalam bahasa pemrograman C :

```
#include <stdio.h>
 main() {
 int i, j, k, l;
 static int M[2][3][2][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2,
                              11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                              31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42 };
 for (i=0;i<2; i++) {
  for(j=0;j<3;j++)   {
   for(k=0; k<2; k++)   {
    for(l=0; l<3; l++)    {
          printf("M[%i][%i][%i][%i] = %i, di %x", i, j, k, l,
                    M[i][j][k][l],
                    &M[i][j][k][l] );
     }
     printf("\n");
    }
    printf("\n");
  }
  printf("\n");
}
getch();
return(0);
}
```

Latihan: Tampilkan alamat M[1, 2, 1, 2]

Contoh susunan multidimensional array, 3 dimensi :



Susunan elemen di memori

untuk x=1 , y dan z jalan : A[1,1,1] A[1,1,2] A[1,1,3]
A[1,2,1] A[1,2,2] A[1,2,3]

untuk x=2 , y dan z jalan : A[2,1,1] A[2,1,2] A[2,1,3]
A[2,2,1] A[2,2,2] A[2,2,3]

untuk x=3 , y dan z jalan : A[3,1,1] A[3,1,2] A[3,1,3]
A[3,2,1] A[3,2,2] A[3,2,3]

Aplikasi Penggunaan Array

Proses Stack (tumpukan) dan Queue (antrian) merupakan alokasi memori dalam bentuk array dimensi satu.

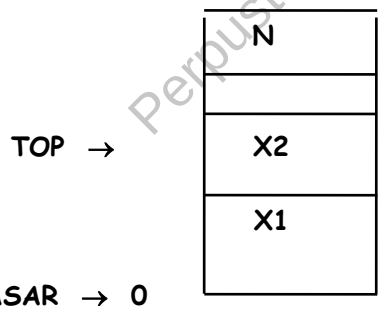
- STACK (tumpukan), berlaku konsep LIFO (Last In First Out)
- QUEUE (antrian), berlaku konsep FIFO (First In First Out) atau FCFS (First Come First Serve)
- DEQUE (Double Ended Queue) yakni antrian yang berlaku untuk 2 arah

Pemilihan dari 3 macam aturan tersebut diatas tergantung kepada masalah yang akan diselesaikan. Apakah perlu memakai queue, stack atau deque.

1. STACK

Adalah suatu list yang semua operasi penambahan (insertion) dan penghapusan (deletion) elemennya dilaksanakan pada satu ujung atas (TOP)

Elemen pertama yang akan dihapus adalah elemen terakhir yang disisipkan, sehingga disebut sebagai "Last In First Out" (LIFO)



Prinsip LIFO :

Data masuk terakhir akan di ambil pertama kali

TOP menunjuk elemen terakhir
Petunjuk awal stack = DASAR

Bila TOP = 0, berarti stack kosong dan TOP berimpit dengan DASAR.

Apabila tumpukan digambarkan dengan array, maka dalam bahasa C dapat dituliskan deklarasinya sebagai berikut :

Tipe_data nama_stack [kapasitas];

Contoh: int s[5]

catatan: stack digambarkan dengan array s

Kondisi stack:

Ada 3 kondisi:

AWAL	KOSONG	PENUH
TOP = 0	TOP = 0	TOP = N

2. Proses yang dapat dilakukan terhadap stack

☑ **PUSH:** menyimpan / memasukkan data kedalam stack (INSERTION)

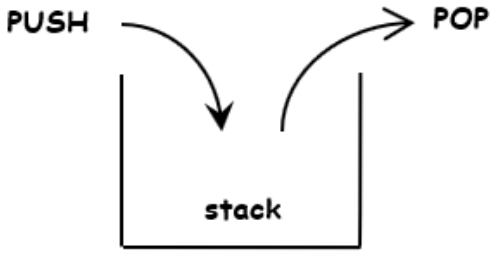
Bila dialokasikan suatu array dengan N elemen yang akan digunakan sebagai stack, maka cara memasukkan data (PUSH) adalah sebagai berikut :

- periksa dulu bahwa TOP < N, kemudian:
- naikkan TOP dengan 1
- isikan data kedalam elemen yang ditunjuk oleh TOP

☑ **POP:** Mengambil data dari stack (DELETION)

Cara pengambilan data adalah sebagai berikut :

- periksa dulu bahwa TOP > 0, kemudian
- copy data dari elemen yang ditunjuk TOP kedalam suatu variabel
- turunkan TOP



Ilustrasi Stack

ALGORITME (dalam bahasa C)

Dinyatakan dalam bentuk fungsi. Dalam C, pemanggilan fungsi dilakukan langsung dengan namanya.

Algoritme dari PUSH dan POP

Aplikasi dalam bahasa pemrograman C:

```
#include <stdio.h>
void push(void);
void pop(void);
int x, top;
int s[5] ,  n = 4;

main() {
  char pilih;  ClrBarloop:
  {
     clrscr();
     gotoxy(25,7);  puts(" Coba Stack");
     gotoxy(25,10); puts(" 1. Push");
     gotoxy(25,12); puts(" 2. Pop");
     gotoxy(25,14); puts(" 3. exit");
     gotoxy(25,17); printf(" Pilih : ");
     scanf("%x", &pilih);

    switch(pilih)   {
    case 1: printf("\n masukkan data x = ");
              scanf("%d", &x);
              push(); getch(); break;
    case 2: pop(); getch(); break;
```

```
   case 3: exit(0); break;
  }
  goto ClrBarloop;
 }
}
void   pop(void) {
 if ( top > 0 )
    { x = s[top];
      printf("\n\r x = %d top = %d" , x, top);
      top = top - 1; }
  else
      printf("stack kosong");
 }
```

Latihan: Buatlah procedure push → void push(void)

3. Aplikasi stack

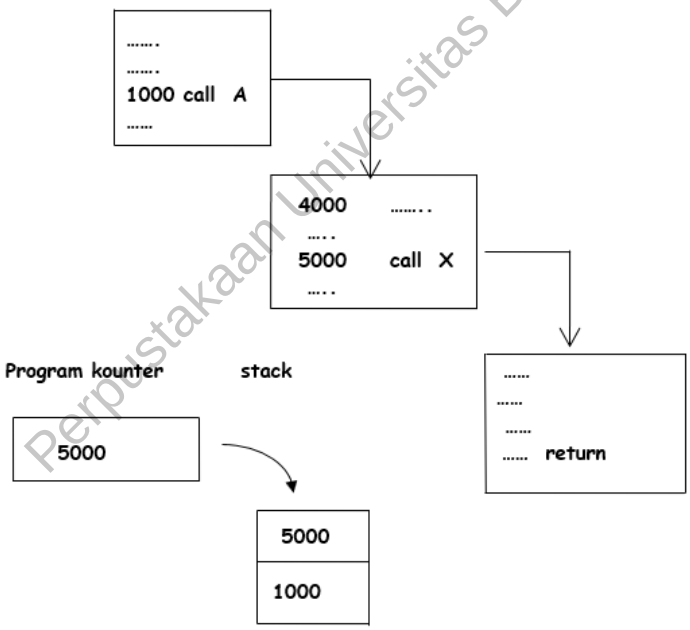
Antara lain:

1. Dalam sistem operasi :
   Untuk penyimpanan return address dalam pemanggilan subroutine.
2. Dalam proses kompilasi untuk memeriksa:
   a. kelengkapan pasangan kurung tutup dan kurung buka

   ((( )))
   PUSH POP

   b. kelengkapan pasangan BEGIN dan END dalam program Bahasa PASCAL atau C

Contoh aplikasi 1:



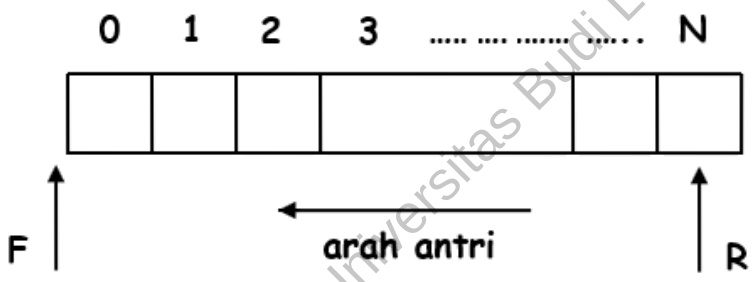
Queue (Antrian)

Prinsip : FIFO (First In First Out)
atau
FCFS (First Come First Serve)
Yang lebih awal masuk akan dilayani terlebih dahulu

Antrian ada 2 macam:

- Linier
- Sirkuler

1. Antrian linier



Menggunakan 2 pointer (penunjuk):

FRONT (F) untuk awal antrian
REAR (R) untuk akhir antrian

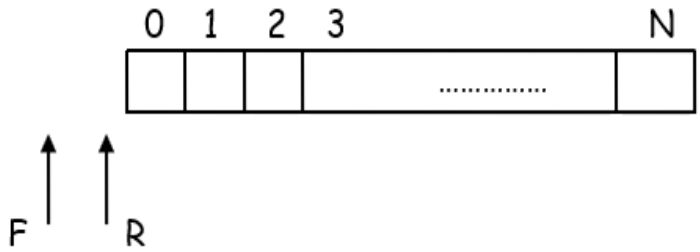
Untuk pengambilan data menggunakan pointer F sedang pemasukan data menggunkana pointer R

SYARAT: F < = R

Deklarasi array 1 dimensi: int Q[5];

2. Beberapa keadaan antrian

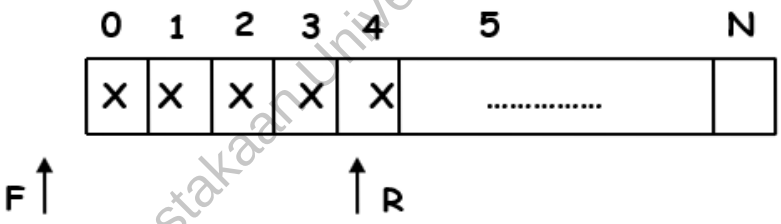
a. Keadaan kosong



Kondisi :

- Keadaan awal, F = 0 , R = 0
- Antrian belum pernah ada. Masih tersedia N tempat.

b. Keadaan masih bisa diisi

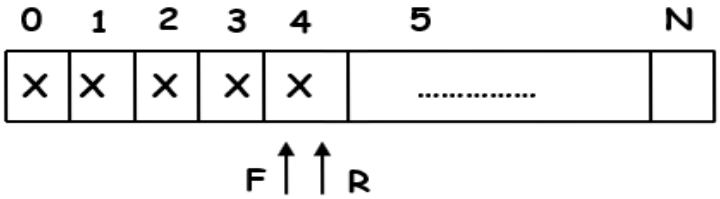


Kondisi :

- Yang antri ada 5 maka → R = 5
- Masih ada tempat → N - 5 ; R < N
- Belum satupun dilayani → F = 0

Bila elemen ke-1 diambil dari antrian, seharusnya elemen ke-1 keluar dari antrian, tapi disini ditunjukkan dengan F yang menunjuk ke elemen **1** → **F = 1**

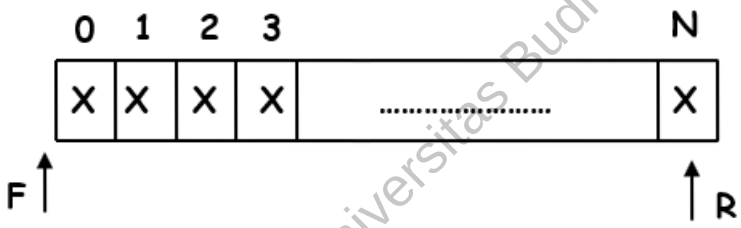
c. Keadaan kosong



Kondisi :

- Keadaan kosong bila F = R atau Front dan Rear berimpit.

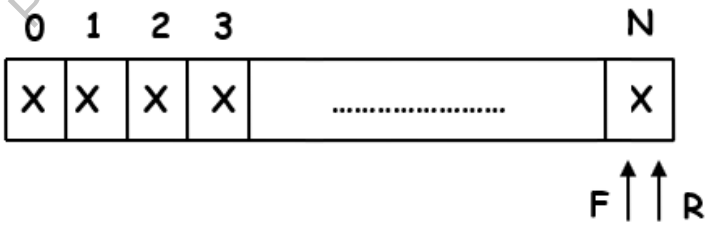
d. Keadaan penuh



Kondisi:

- Keadaan penuh yang sebenarnya → F = 0, R = N
- Semua elemen array (S) terisi.

e. Kondisi Khusus: PENUH TAPI KOSONG



Penuh bukan berarti semua terisi.
Pada kondisi F = R = N; tidak mungkin menambah atau mengambil data karena akan menyalahi persyaratan FIFO dan F <= R.
Apabila F = N dan R = N, antrian harus dikembalikan kekeadaan awal (di RESET)

- F dan R kembali keawal lagi yaitu F = R = 0.
- Kondisi ini disebut Penuh Tapi Kosong.

KESIMPULAN:

Syarat yang harus dipenuhi	:	F <= R
Kondisi awal	:	F = R = 0
Kosong	:	F = R
Penuh	:	R = N
Harus di reset bila	:	F = R = N

3. Operasi antrian:

1. Penambahan data kedalam antrian (Insert)
Langkah:
- memastikan bahwa antrian belum penuh
- menggeser maju R satu langkah
- mengcopy data ke elemen antrian Q yang ditunjuk oleh R.

2. Pengambilan data dari antrian (Delete)
Langkah:
- memastikan bahwa antrian ada isinya
- menggeser F maju satu langkah
- mengcopy isi elemen yang ditunjuk F ke variabel data

3. Kembali ke kondisi awal antrian
Langkah : - kembalikan R dan F ke posisi awal

Program antrian dalam bahasa C tanpa penggeseran data

```
#include <stdio.h>
#include <bios.h>
#include <conio.h>
#include <process.h>
#define n 5
```

```
void insert( void);
void delete( void );
int  r = 0, f = 0, q[n+1], x;

main(){
  int  pilih;
  ClrBarloop:
  clrscr();
  gotoxy(25,7);  puts("    antrian");
  gotoxy(25,10); puts("1.  insert");
  gotoxy(25,12); puts("2.  delete");
  gotoxy(25,14); puts("3.  reset");
  gotoxy(25,16); puts("4.  quit");
  gotoxy(25,19); printf("    pilih  :    ");
  scanf("%d",  &pilih);
  switch(pilih) {
  case  1: printf("\r\nMasukkan  data  x:   ");
           scanf("%d", &x);
           insert();
           printf("\r\n");
           getch();
           break;
  case  2: delete();
           printf("\r\n");
           getch();
           break;
  case  3: f = 0;
            r = 0;
           printf("\r\n f = %d  r = %d ",f, r);
           getch();
           break;
  case  4: exit(0);
           break;
  }
   goto  ClrBarloop;
}

void insert( void){
   if ( r <  n ) {
      r = r + 1;
      q[r] = x;
```

```
    printf(" r = %d    x = %d  ",  r,x);
 }
 else
   printf("antrian  penuh");
}
```

Latihan: Buat procedure → void delete(void)

4. Contoh Aplikasi
   - Misalkan suatu antrian dari bis angkutan kota perlu dikelola, dimana setiap bis diberi identitas berupa nomor.
   - Setiap kali bis masuk terminal, bis tersebut harus antri menunggu penumpang. Bis diberangkatkan jika penuh. Panjang antrian dibatasi untuk 20 buah bis.

Penyelesaian:

1. Definisikan antrian tersebut
2. Tuliskan Procedure kedatangan (DATANG) dan keberangkatan (BERANGKAT) dan menggeser antrian ke depan (GESER) bila ada bis yang berangkat meninggalkan antrian.

Penyelesaian dalam algoritme:

1. Deklarasikan konstanta yang menyatakan batas dari array.

```
#define   N   20
int   BIS[N] ,  F,  R  ;
```

atau array langsung dibatasi dengan suatu konstanta

```
int   BIS[20],  F,  R;
```

2. Lengkapi program ini dengan procedure datang dan geser.

```
#include  <stdio.h>
#include <bios.h>
#include <conio.h>
#include <process.h>
#define  N   5

void datang( void);
void berangkat( void ),  geser(void);
void initial(void);

int   R, F,  BIS[N+1], NOBIS;
int   I;

main(){
  int  pilih;
  initial();
  ClrBarloop:
  clrscr();

  gotoxy(25,7);   puts("antrian");
  gotoxy(25,10);  puts("1.  datang");
  gotoxy(25,12);  puts("2.  berangkat");
  gotoxy(25,14);  puts("3.  quit");
  gotoxy(25,17);  printf("    pilih  :    ");

  scanf("%d",  &pilih);
          switch(pilih)  {
                  case  1:
                          printf("\r\nMasukkan  NOBIS:   ");
                          scanf("%d", &NOBIS);
```

```
                        datang();
                        printf("\r\n");
                        getch();
                        break;
                Case  2:
                        berangkat();
                        printf("\r\n");
                        getch();
                        break;
                 case  3:
                        exit(0);
                        break;
        }
    goto  ClrBarloop;
}

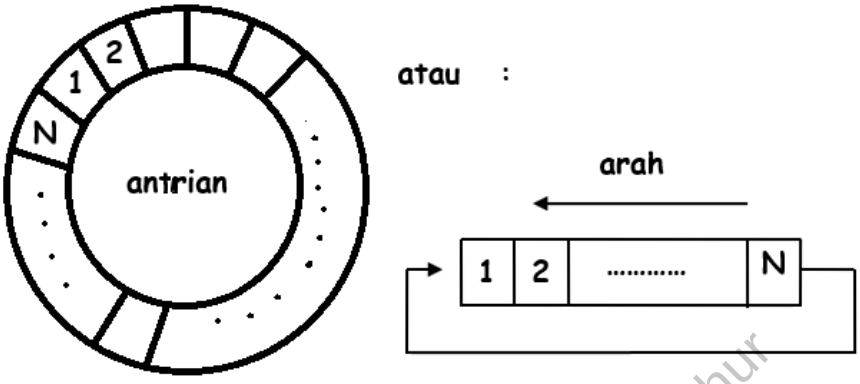
void initial(void)
{
   F = 0;
   R = 0;
}

void berangkat(void)
{
  if  ( F < R )  {
       F = 1;
       NOBIS = BIS[F];
       geser();
       printf(" NOBIS = %d  ", NOBIS);
    }
  else  {
      printf("antrian  kosong");
   }
}

Void datang ( void )
{  ....

}
```

Circular Queue (Antrian Melingkar)



1. Prinsip tetap FIFO

Arah antrian tetap sama dengan antrian linier biasa, hanya saja setelah sampai ke elemen yang ke n akan kembali ke elemen 1 tanpa harus di RESET

Karena aliran melingkar, F dan R dapat saling menyusul sehingga:

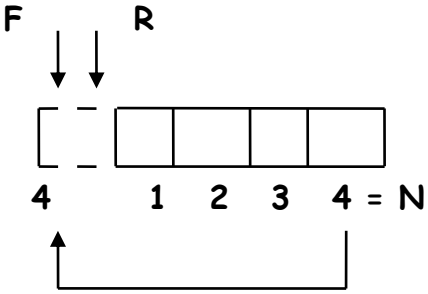
F TIDAK SELALU <= R

Perbandingan kondisi antrian antara linear queue dan circular queue:

KONDISI	ANTRIAN LINIER	ANTRIAN SIRKULER	KETERANGAN
Keadaan awal	F = 0 R = 0	F = N R = N Flag off /not isi	belum pernah diisi
Keadaan kosong	F = R	F = R Flag off/not isi	semua isi sudah diambil
Keadaan Penuh	R = N	F = R Flag on/ isi	tak bisa diisi lagi

Contoh beberapa kondisi yang terjadi di antrian sirkuler:

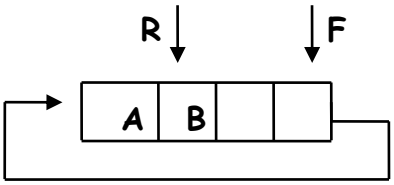
1.



Bila N = 4 maka kondisi inisial :

F = 4
R = 4
Flag off atau not isi

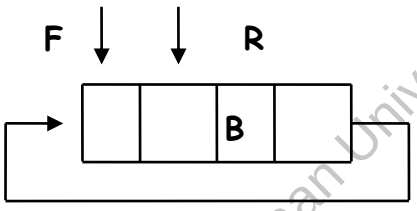
2



insert A dan B

R = 2
F = N = 4
Flag on atau isi

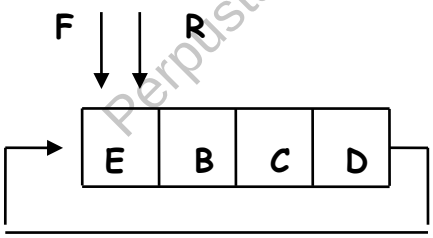
3.



delete A

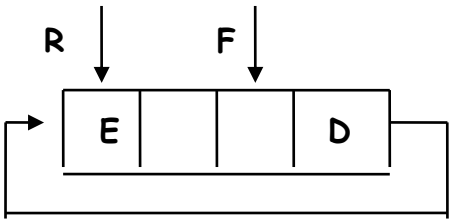
F = 1
R = 2
Flag on

4.



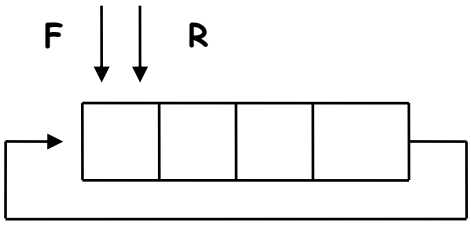
insert C, D dan E
antrian penuh
F = R
Flag on

5.



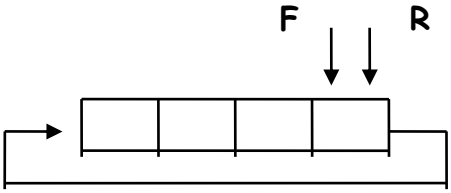
delete B dan C
F = 3
R = 1
Flag on

6.



delete D dan E
antrian kosong F = R
Flag off atau not isi

7



RESET

kondisi awal
F = R = N

Terlihat dari urutan proses diatas, nilai F bisa lebih kecil atau lebih besar dari R, dimana ini merupakan hal yang tidak bisa terjadi di antrian linier (linear queue). Oleh karena itu agar antrian dapat digunakan secara MAKSIMAL harus diwujudkan dalam bentuk SIRKULER.

Proses Insert :

- periksa apakah antrian tidak penuh (F <> R) atau Flag = off
- R maju satu langkah
- isi array, elemen yang ke R
- isikan FLAG = 'ON'

Proses Delete :

- Periksa apakah Flag = on, bila tidak → antrian kosong
- F maju satu langkah
- ambil isi array, elemen yang ditunjuk oleh F
- periksa lagi, bila antrian menjadi kosong setelah proses delete (F = R) buat Flag = off

2. Program Queue dalam bahasa C

```
#include  <stdio.h>
#include <bios.h>
#include <conio.h>
#include <process.h>

void insert( void );
void delete( void );

int  r =4 , f = 4, q[5], x , i ;
char flag = '0';
int n=4;

main() {
   int  pilih;
   ClrBarloop:
   clrscr();
   gotoxy(25,7);puts("antrian");
   gotoxy(25,10);puts("1.  insert");
   gotoxy(25,12);puts("2.  delete");
   gotoxy(25,14);puts("3.  quit");
   gotoxy(25,17);printf("    pilih :    ");
   scanf("%d", &pilih);
   switch(pilih) {
    case  1:
          printf("\r\nMasukkan  data  x:  ");
          scanf("%d", &x);
          insert();        printf("\r\n");
          getch();
          break;
   case  2:
          delete();
          printf("\r\n");
          getch();
          break;
  case  3:
          exit(0);
          break;
```

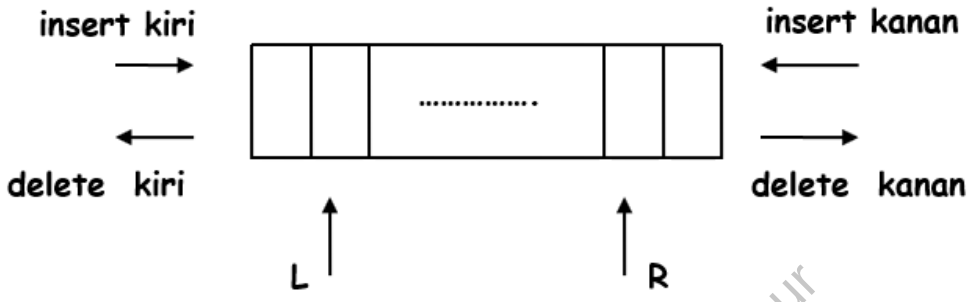
```
  }
  goto  ClrBarloop;
}

void insert( void ) {
   if (( f !=  r ) ||  ( flag == '0'))  {
         r = ( r %  n) + 1;
         q[r] = x;
         flag = '1';
          printf(" r = %d    x = %d  ",  r,x );
       }
 else
       printf("antrian  penuh"  );
}
```

Latihan: Buat Procedure DELETE dari antrian sirkuler

Deque (Double-Ended Queue)

Ilustrasi Qeque



1. Operasi Deque:
   - **a.** Penambahan data (Insert data)
     1) dari kiri
     2) dari kanan

   - **b.** Pengambilan data (Delete data)
     1) dari kiri
     2) dari kanan

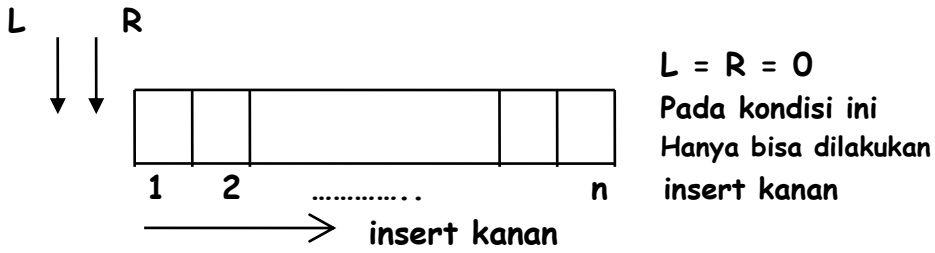
Sebagai basis tetap menggunakan array, dengan melihat mana batas paling kiri (LEFT disingkat L) dan mana batas paling kanan (RIGHT disingkat R).

SYARAT: L tidak boleh mendahului **R**

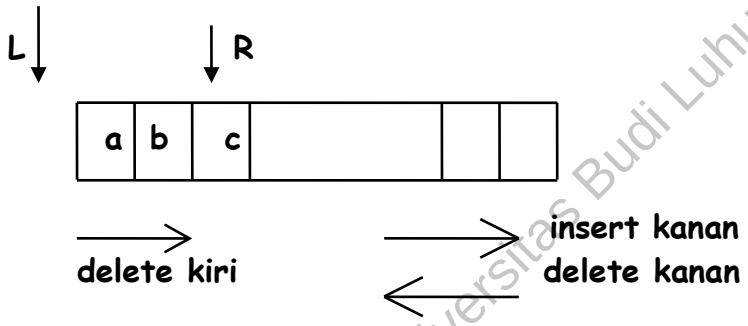
L <= R

Cara menempatkan L dan R:

1. **Kondisi awal (inisial)**



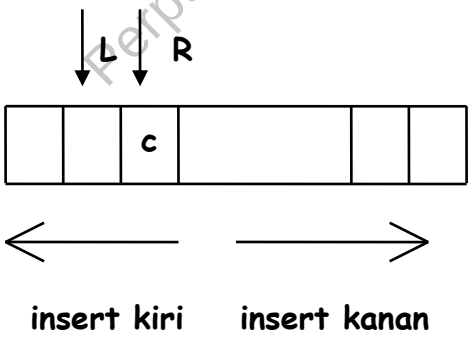
2. **Insert kanan sebanyak 3 elemen**



Pada kondisi:

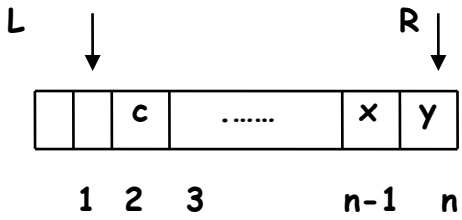
L = 0 dan R = 3, tidak bisa diinsert dari kiri dan bisa di delete dari kiri atau kanan

3. **Delete kiri 2 elemen**



L = 2
R = 3
bisa di insert dari kiri atau kanan
bisa di ambil dari kiri atau kanan

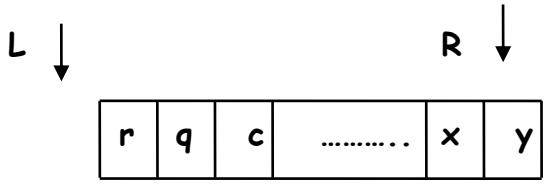
4. **Insert kanan sampai n**



L = 2
R = n (penuh kanan)

tidak bisa di insert dari kanan

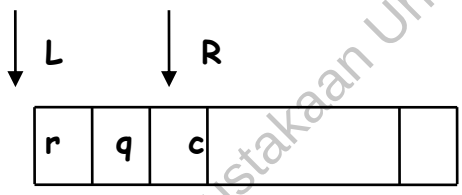
5. Insert kiri 2 elemen



L = 0 (penuh kiri)
R = n (penuh kanan)

Dalam keadaan seperti gambar diatas antrian tidak bisa diisi baik dari kiri maupun dari kanan. Kalau mengambil dapat bebas.

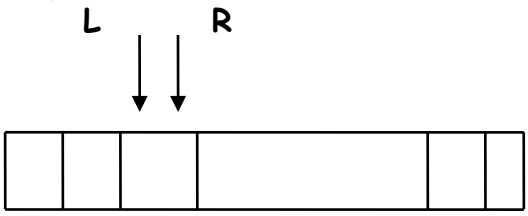
6. Delete kanan sampai R = 3



L = 0
R = 3

tidak bisa diisi dari kiri

7. Delete kiri 3 elemen



L = R

Bila L = R diposisi mana saja maka antrian dalam kondisi kosong.

Kesimpulan:

AWAL	**L = R = 0**	
KOSONG	**L = R**	
PENUH	**L = 0**	**penuh kiri**
	R = n	**penuh kanan**
	L = n & R = n	**penuh sebenarnya**

2. Program Deque dalam C

```
#include  <stdio.h>
#include <bios.h>
#include <conio.h>
#include <process.h>

void insert_kiri( void);
void delete_kanan( void );
void insert_kanan( void);
void delete_kiri( void );

int l = 0 , r = 0 ,  d[5],  x;
int n = 4;

main(){
   int  pilih;
   ClrBarloop:
   clrscr();
   gotoxy(25,5);puts("antrian");
   gotoxy(25,8);puts("1.  insert_kiri");
   gotoxy(25,10);puts("2.  delete_kanan");
   gotoxy(25,12);puts("3.  insert_kanan");
   gotoxy(25,14);puts("4.  delete_kiri");
   gotoxy(25,16);puts("5.  quit");
   gotoxy(25,19);printf("    pilih  :    ");
   scanf("%d",  &pilih);
```

```
  switch(pilih)  {
  case  1:
         printf("\r\nMasukkan data x:  ");
         scanf("%d", &x);
         insert_kiri();
          printf("\r\n");
         getch();
         break;
  case  2:
         delete_kanan();
          printf("\r\n");
         getch();
         break;
  case  3:
         printf("\r\nMasukkan data x:  ");
         scanf("%d", &x);
         insert_kanan();
         printf("\r\n");
         getch();
         break;
  case  4:
         delete_kiri();
          printf("\r\n");
         getch();
         break;
 case  5:
         exit(0);
         break;
 }
 goto  ClrBarloop;
}

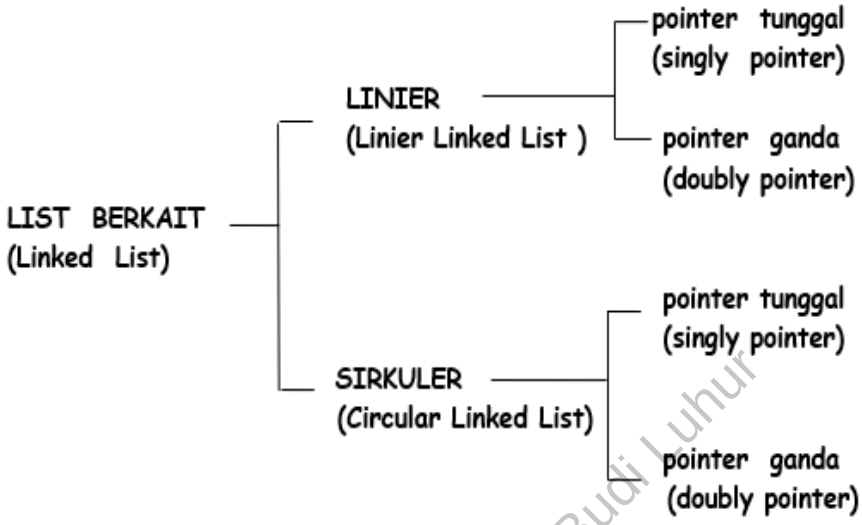
void insert_kiri( void) {
  if ( l > 0 )  {
  d[l] = x;
  printf(" l = %d    x = %d  ",  l,x);
    l = l - 1;
```

```
  }
  else
    printf("antrian  penuh");
}

void   delete_kanan(void) {
  if  ( r > l )  {
    x = d[r];
    printf(" r = %d    x = %d  ",  r, x);
    r = r - 1;
   }
    else {
       printf("antrian  kosong");
  }
}
```

Latihan: Buat procedure untuk insert_ kanan dan delete_ kiri

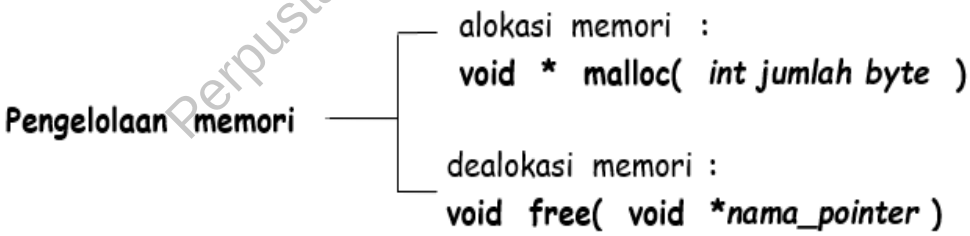
List Berkait ("Linked List")



1. Linked List

Pengelolaan memory secara dinamis artinya tidak perlu mengalokasi memory lebih awal secara tetap (fixed) Dengan cara dinamis dapat dilakukan:

- alokasi dan
- dealokasi memory.



contoh :

```
char *ptr;
ptr = (char *) malloc( 500 * sizeof(char));
free(ptr);
```

2. Linier Singly Linked List

Satu elemen terdiri dari 2 bagian utama :

a. bagian / field yang menyimpan data

b. bagian / field yang menyimpan alamat record_next

data	**alamat rec_next**

Dalam C di definisikan sebagai berikut:

```
struct    nama_struktur_rec
          {  tipe_data1     nama_field1 ;
             ……….     ……….. ;
             tipe_data2    nama_field_2;
             struct nama_struc_rec  *nama_pointer };
```

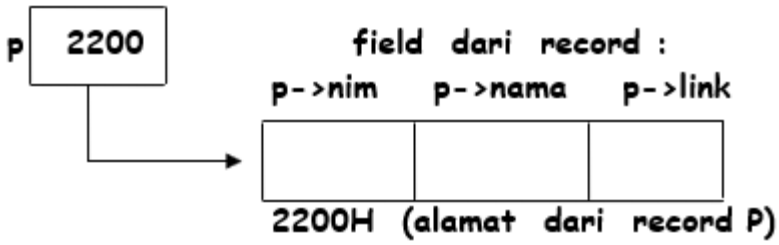
contoh:

```
struct simpul
      {  unsigned int nim;
         char nama[10];
         struct simpul  *link;
      };
```

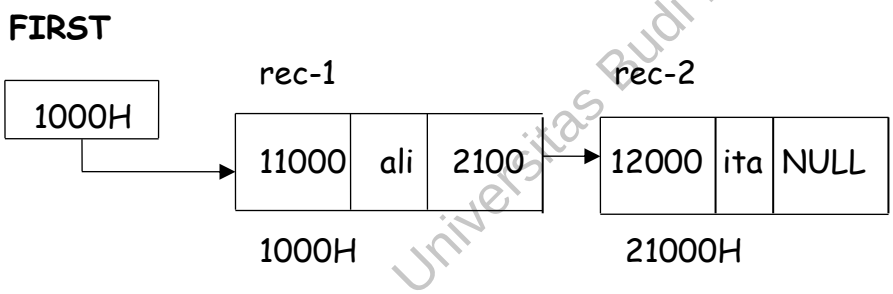
Bila akan membentuk simpul (record) P maka instruksinya adalah:

```
struct  simpul  p;        /*  deklarasi  pointer  p  */

/*  bentuk  record  baru  */
 p = (struct  simpul  *) malloc(sizeof(struct  simpul));
```



Dalam operasi linked list selalu ada pointer yang dipakai sebagai identitas awal dari suatu linked list. Misal dalam contoh dibawah ini digunakan pointer FIRST sebagai identitas awal dari suatu linked list.



TAHAP CREATE, PENAMBAHAN DAN PENGHAPUSAN

1. PENAMBAHAN dan CREATE

Langkah-langkah secara umum :

a. menciptakan elemen baru
b. mengisi data ke elemen tersebut
c. cek kondisi pointer_awal dari linked list,
   bila :
   - **kosong**, lakukan proses cre**ate**
   - **not kosong,** lakukan proses **penambahan awal, penambahan tengah, penambahan akhir**

d. menempatkan posisi baru dari pointer awal, bila penambahan record baru berupa **penambahan awal** atau **create**.

- Membuat identitas baru dari record akhir dengan mengisi LINK nya dengan NULL, bila penambahannya berupa **penambahan akhir**.
- Membuat sambungan baru ke linked list, bila penambahannya berupa **penambahan tengah**.

2. PENGHAPUSAN

Langkah-langkah secara umum :

1. Cek kondisi not kosong dari linked list,
   bila ya, lakukan langkah penghapusan
   bila tidak berikan pesan bahwa linked list kosong
2. Set pointer ke record yang akan di delete
   Bila record yang akan didelete jauh dari record-1, maka lakukan penelusuran untuk menempatkan pointer ke record yang akan di delete.
3. Untuk hapus :

   AWAL → pindah pointer_awal ke record-1 baru
   TENGAH → lakukan sambungan baru
   AKHIR → isikan NULL ke link yang akan menjadi record akhir baru.
4. keluarkan isi field data dari record yang akan didelete (option). Hapus record yang akan di delete.

3. Aplikasi Linier Singly Linked List dalam bahasa C

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

struct  simpul
{   char nim[11];
    char nama[20];
    struct simpul *link;
```

```
};

struct simpul  *first = NULL;
struct simpul  *penunjuk;
char nim1[11];
char nama1[20];
void  insert_awal(void);
void  insert_tengah(void);
void  insert_akhir(void);
void  delete_awal(void);
void  delete_tengah(void);
void  delete_akhir();
void  tampil(void);

main() {
        char pilih;
        ClrBarloop:
        {
        clrscr();
        gotoxy(20,5); puts("Linier singly Linked list");
        gotoxy(25,7); puts("1.  insert");
        gotoxy(25,9); puts("2.  delete");
        gotoxy(25,11);puts("3.  tampilan");
        gotoxy(25,13);puts("4.  exit");
        gotoxy(25,16);printf("Pilih : ");
        scanf("%x", &pilih);
        switch(pilih)           {
        case  1:
           printf("\n        masukkan  data NIM  = ");
           scanf("%s", &nim1);
           printf("\n        masukkan  data NAMA = ");
           scanf("%s", &nama1);
           if (first == NULL)
                {
                  insert_awal();
                }
           else
```

```
            {  penunjuk  =  first;
               if (atof(nim1) < atof(penunjuk->nim))
                  {
                     insert_awal();
                  }
              else
                 { while (penunjuk->link != NULL)
                          {penunjuk = penunjuk->link;}
                     if ( atof(nim1) > atof(penunjuk->nim))
                          {
                              insert_akhir();
                           }
                     else
                          insert_tengah();
                  }
              }
        getch();
        break;

case  2: printf("\n          NIM yang akan di delete  :  ");
          scanf("%s", &nim1);
        if (first != NULL)
          {  penunjuk  =  first;
             if (atof(nim1) == atof(penunjuk->nim))
                {
                   delete_awal();
                }
             else
               {  while (penunjuk->link != NULL)
                     {  penunjuk = penunjuk->link; }
          if (atof(nim1) == atof(penunjuk->nim))
                     {  delete_akhir();}
          else
              { penunjuk = first->link;
              while(atof(penunjuk->nim) != atof(nim1))
                  {  penunjuk = penunjuk->link;
                     if(penunjuk->link == NULL)
```

```
                        {
                         printf("\n      nim = %s  tidak ada di file", nim1);
                        getch();
                        goto ClrBarloop;}
                         }
                delete_tengah();
              }
            }
          }
          getch();
          break;

  case   3: tampil();
            getch();
            break;

  case  4: exit(0);
            break;
  }
  goto ClrBarloop;
 }
}

void   insert_awal(void) {
  struct  simpul  *p;
  p = (struct  simpul *) malloc(sizeof(struct simpul));
  strcpy(p->nim, nim1);
  strcpy(p->nama,  nama1);

  if (first != NULL)
    { p->link = first;
      first = p;
      printf("\n              sisip awal");
     }
  else
   { p->link = NULL;
     first = p;
```

```
     printf("\n              create  file");
    }
}

void   insert_tengah(void) {
  struct  simpul  *p,  *q, *k;
  p = (struct  simpul *) malloc(sizeof(struct simpul));
  strcpy(p->nim, nim1);
  strcpy(p->nama, nama1);

  if (first !=  NULL)
    {   q = first;
       while (atof(q->nim)   <   atof(nim1) )
            {
                k = q;
                q = q->link;
             }
    p->link = q;
    k->link = p;
    printf("\n              sisip tengah");
   }
   else
   {
       insert_awal();
    }
}

void   tampil(void) {
 struct  simpul  *p;
 if (first != NULL)
   {  p = first;
      while ( p != NULL)
             {  strcpy(nim1, p->nim);
                strcpy(nama1, p->nama);
                p = p->link;
                printf("\n   nim  =  %s        nama = %s", nim1, nama1);  }
 }
```

```
}

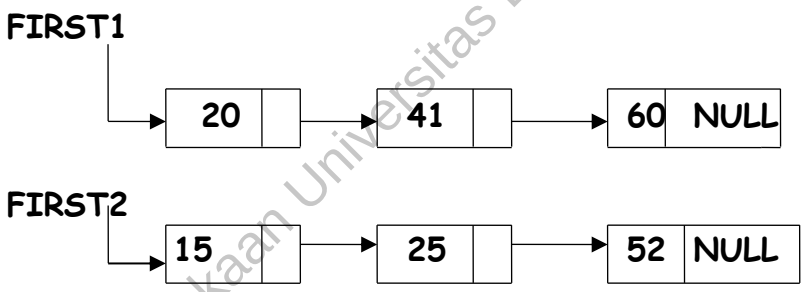
void delete_awal(void) {
 struct  simpul  *p;
 if (first != NULL)
 {
   p = first;
   first = first->link;
   strcpy(nama1, p->nama);
   p->link = NULL;
   printf("\n            nim =  %s   nama = %s  ", nim1, nama1);
   free(p);
 }
  else  printf("\n list kosong");
}

void delete_akhir(void) {
 struct  simpul  *p,  *q;
 if (first != NULL)
 {
   p = first;
   while (p->link  !=  NULL)
   {
           q = p;
           p = p->link;
   }
   q->link = NULL;
   strcpy(nim1, p->nim);
   strcpy(nama1, p->nama);
   printf("\n            nim =  %s   nama = %s  ", nim1, nama1);
   p->link = NULL;
   free(p);
 }
  else  printf("\n list kosong");
}
```

Latihan:

1. Buat procedure insert_akhir, delete_tengah

2. Gabungkan 2 linked list dibawah ini dalam kondisi tetap sort.



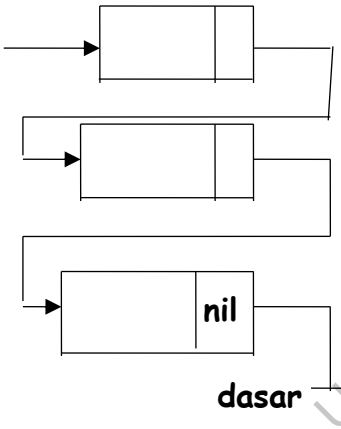
Aplikasi Linier Singly Linked List

- LINKED STACK
- LINKED QUEUE

1. Linked stack

   Prinsip: tetap berlaku prinsip LIFO

STACK DYNAMIS:



Kondisi kosong: TOP = NIL

Kondisi penuh : TOP = tak ter-definisi

deklarasi:

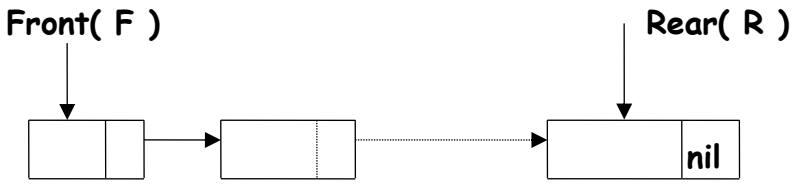
```
struc  simpul
  {
      struct  simpul  *link;
  };
struct simpul *top =  NULL;
```

Langkah-langkah procedure PUSH sama dengan langkah-langkah pada insert_awal dan create dari linked list.
Langkah-langkah procedure POP sama dengan langkah-langkah procedure delete_awal.

Latihan: Buat procedure POP dan PUSH

2. Linked queue

Prinsip: tetap berlaku prinsip FIFO



Insert suatu simpul dilakukan pada Rear (R) sedang untuk delete dilakukan pada Front (F) sehingga tetap berlaku prinsip FIFO.

Kondisi kosong terjadi bila: R = NIL dan
F = NIL

Procedure Insert langkahnya sama dengan procedure Sisip Awal dan Create dari Linier Linked List.

Latihan: Buat procedure Delete dari linked Queue

3. List berangkai melingkar (circular singly linked list)

Dapat digunakan 2 macam versi, yaitu:

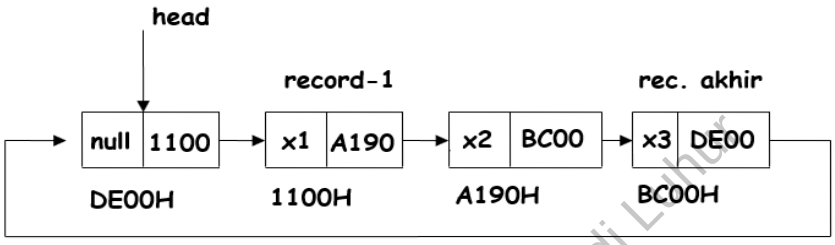
1. menggunakan simpul kepala yang fungsinya hanya sebagai penghubung.
2. tanpa menggunakan simpul kepala

ad.1

Pada linked list ini, field info dari simpul *head* berisi nil jadi bukan suatu data.

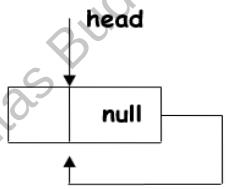
Simpul head bukan record-1

Record pertama adalah head->link.

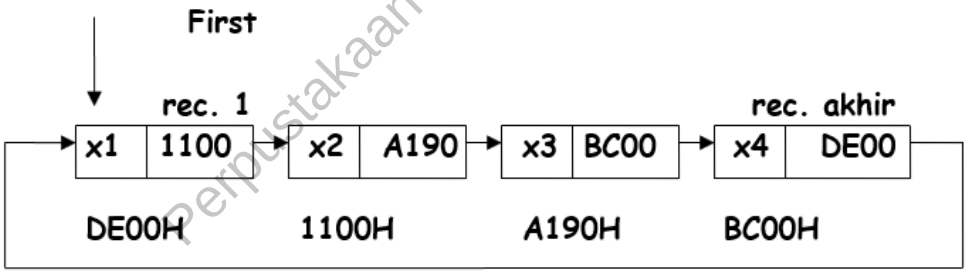


Kondisi kosong terjadi bila :

head->link = head



ad.2



Record pertama adalah simpul First. Kondisi kosong terjadi bila:

First = NULL

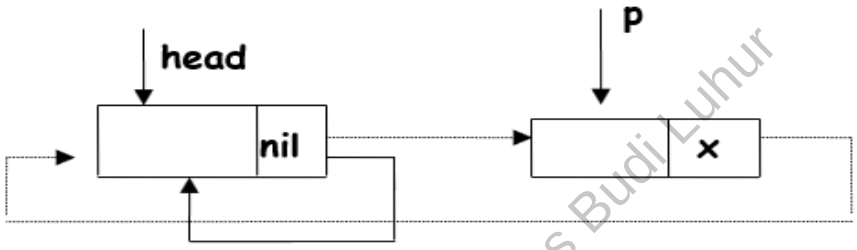
Pada pembahasan selanjutnya digunakan versi 1 dalam memberitahu gambaran operasi Penambahan dan Penghapusan pada list berangkai tunggal melingkar :

PENAMBAHAN

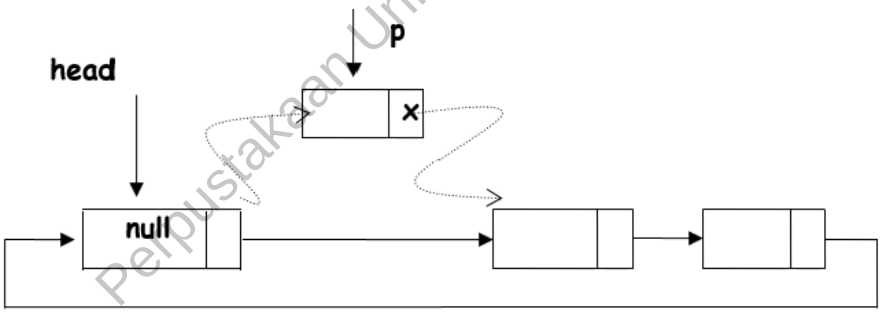
Langkah-langkah umum tetap sama dengan linier singly linked list.

Insert_awal dan Create

Create :



Sisip Awal :



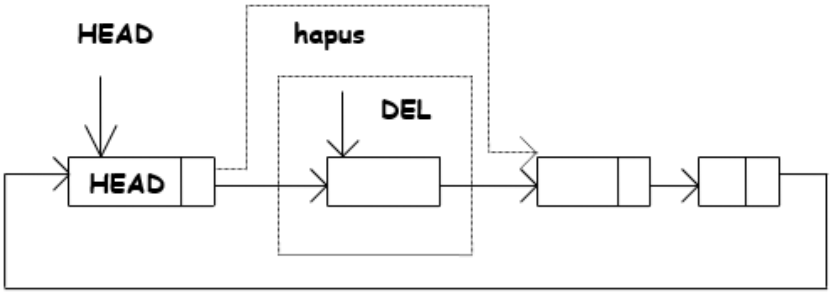
Langkah-langkah untuk insert_akhir dan insert_tengah lihat logika pada linier singly linked list.

PENGHAPUSAN

Pada list melingkar (Circular List) elemen HEAD tidak bisa dihapus.

Langkah-langkah sama dengan linier singly linked list.

Hapus Awal



Langkah-langkah untuk delete_akhir dan delete_tengah sama dengan linier singly linked list

Program aplikasi circular singly linked list dengan bahasa C:

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

struct  simpul {
   char  nim[11];
   char nama[10];
   struct simpul *link;
};

struct  simpul  *head = NULL;
struct  simpul  *penunjuk;
char  nim1[11];
char nama1[10];

void  init(void);
void  insert_awal(void);
void  insert_tengah(void);
void  insert_akhir(void);
void  delete_awal(void);
```

```
void  delete_tengah(void);
void  delete_akhir();
void  tampil(void);

main() {
   char pilih;
    init();
   ClrBarloop:
   {
   clrscr();
   gotoxy(20,5); puts("Circular singly Linked list");
   gotoxy(25,7); puts("1.  insert");
   gotoxy(25,9); puts("2.  delete");
   gotoxy(25,11); puts("3.  tampilan");
   gotoxy(25,13); puts("4.  exit");
   gotoxy(25,16); printf("Pilih : ");
   scanf("%x",  &pilih);

   switch(pilih)  {
    case  1:   printf("\n    masukkan  data NIM  = ");
               scanf("%s", &nim1);
               printf("\n      masukkan  data NAMA = ");
               scanf("%s", &nama1);
              if (head->link == head)  {
                  insert_awal(); }
              else   {
                  penunjuk  =  head->link;
                  if (atof(nim1)  < atof(penunjuk->nim))   {
                      insert_awal();  }
                  else
                     { while (penunjuk->link != head)
                              {penunjuk = penunjuk->link;}
                        if (atof(nim1) > atof(penunjuk->nim))
                              {  insert_akhir();}
                       else  {  insert_tengah();}
                       }
                  }
```

```
 getch();
 break;

case  2:   printf("\n       NIM yang akan di delete  :  ");
           scanf("%s", &nim1);
           if ( head->link != head)
             {  penunjuk  =  head->link;
               if ( atof(nim1) == atof(penunjuk->nim))
                     {delete_awal();}
               else
                    { while (penunjuk->link != head)
                              { penunjuk = penunjuk->link; }
                      if(atof(nim1)== atof(penunjuk->nim))
                              {delete_akhir();}
                     else
                       {  penunjuk = head->link;
                         while(atof(penunjuk->nim) != atof(nim1))
                              {  penunjuk = penunjuk->link;
                                 if(penunjuk->link == head)
                            { printf("\n     nim = %s tidak ada di file", nim1);
                              getch();
                              goto ClrBarloop;}
                         }
                     delete_tengah();
                  }
             }
      }
getch();
break;

case  3: tampil();
         getch();
         break;

case  4: exit(0);
         break;
}
```

```
 goto ClrBarloop;
 }
}

void   init(void) {
   struct   simpul  *p;
   p = (struct simpul *) malloc(sizeof(struct simpul));
   strcpy(p->nim , 00000);
   strcpy(p->nama , "head");
   head  = p;
   p->link = p;
 }

 void   insert_awal(void){
  struct  simpul  *p;
  p = (struct  simpul *) malloc(sizeof(struct simpul));
  strcpy(p->nim , nim1);
  strcpy(p->nama, nama1);

  if (head->link  !=  head)
  { p->link = head->link;
     head->link = p;
     printf("\n              sisip awal");
    }
    else
    {
      head->link =  p ;
      p->link =  head ;
      printf("\n              create  file");
    }
 }

void   insert_tengah(void) {
  struct  simpul  *p,  *q, *k;
  p = (struct  simpul *) malloc(sizeof(struct simpul));
  strcpy(p->nim , nim1);
  strcpy(p->nama, nama1);
```

```
  if (head->link  !=  head)
    { q = head->link;
      while (atof(q->nim) < atof(nim1))
              { k = q;
                q = q->link; }
    p->link = q;
    k->link = p;
    printf("\n              sisip tengah");     }
else
   { head->link = p;
    p->link = head ;
    printf("\n  create  file");}
 }

/*  Buat  procedure  insert_akhir  disini ……..*/

 void   tampil(void) {
  struct  simpul  *p;
  if ( head->link != head)
   { p = head->link;
     while ( p  != head )
             { strcpy(nim1 , p->nim);
               strcpy(nama1, p->nama);
              p = p->link;
              printf("\n     nim  =  %s     nama1 = %s", nim1, nama1);    }
    }
 }

/*Buat procedure delete_awal……..*/

 void delete_tengah(void) {
  struct  simpul  *p,  *q;
  if (head->link != head)
   { p = head->link;
     while (atof(p->nim)  !=  atof(nim1))
             { q = p;
               p = p->link; }
```

```
    q->link  =  p->link;
    printf("\n        nim =  %s    nama = %s  ", nim1, nama1);
    free(p);
   }
 else  printf("\n      list kosong");
}

void delete_akhir(void){
 struct  simpul  *p,  *q;

 if (head->link != head)
   { p = head->link;

     while (p->link  !=  head)
            {q = p;
             p = p->link;   }
     q->link = head;
     strcpy(nim1 , p->nim);
     strcpy(nama1, p->nama);
     printf("\n        nim =  %s   nama = %s  ", nim1, nama1);
     free(p);
    }
  else  printf("\n    list kosong");
}
```

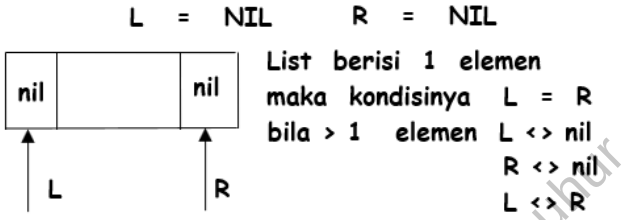
Latihan: Buat procedure insert_akhir dan delete_awal

Linier Doubly Linked List

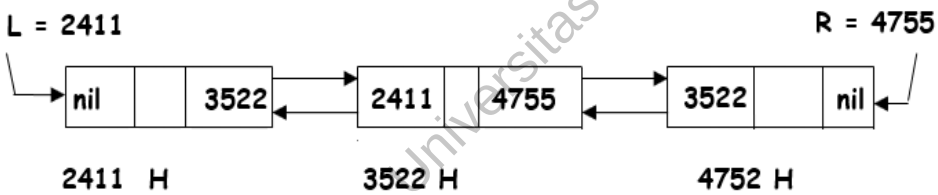
Disebut juga dengan two way chain

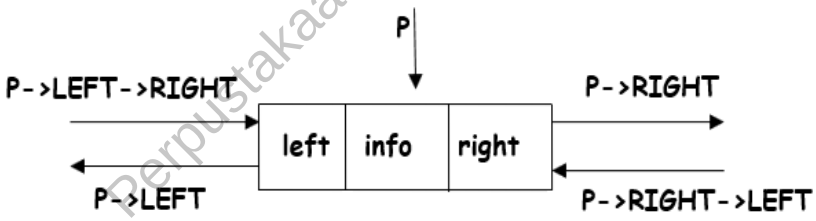
Deklarasi:

Kondisi kosong :



Kondisi isi :





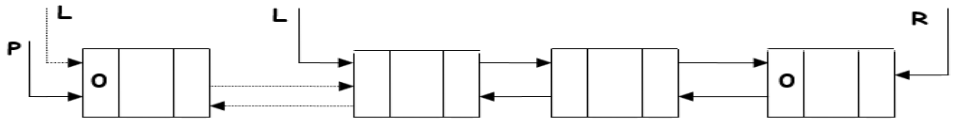
Operasi yang bisa terjadi pada linier doubly linked list adalah

Insert :	insert_ kiri	delete :	delete_kiri
	insert_kanan		delete_kanan
	insert_ tengah		delete_tengah

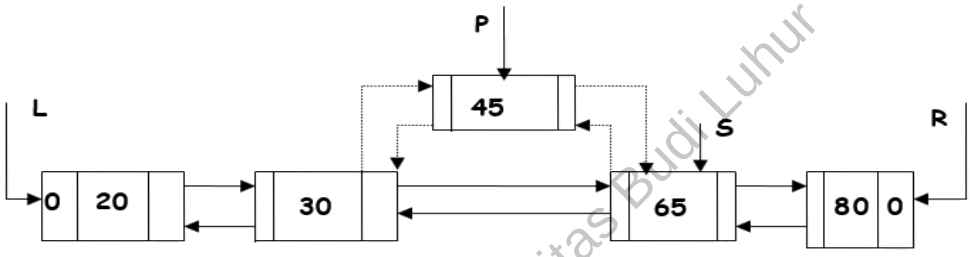
1. Penambahan / Insert

a. Insert_kiri

Urutan langkah-langkah lihat insert_awal linier singly linked list.



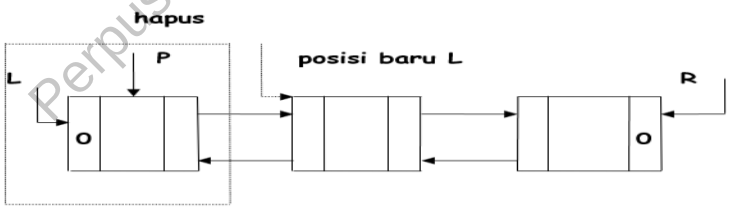
b. Insert_Tengah



2. Penghapusan / Delete

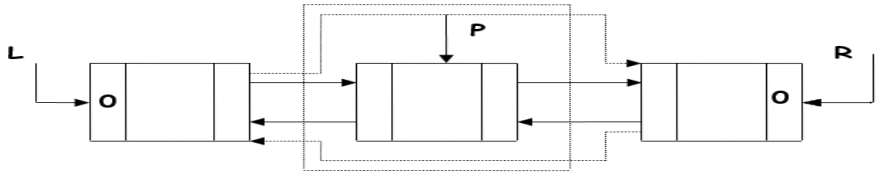
a. Delete - Kiri

Langkah - langkah seperti penghapusan pada linked list yang lain.



b. Delete_Tengah

Semua operasi hapus tengah selalu X sebagai masukan, bukan sebagai keluaran (output) seperti operasi hapus lainnya.



Struktur simpul yang akan dipakai sebagai contoh adalah sebagai berikut:

```
struct simpul
        {  char  nim[11];
           char nama[10];
           struct simpul *left;
           struct simpul *right;
         };
```

Kondisi awal dari pointer L dan pointer R dinyatakan seperti dibawah ini:

```
struct simpul *L= NULL;
struct simpul *R = NULL;
```

3. Prosedur–prosedur insert dan delete dalam C

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

struct  simpul {
  char nim[11];
  char nama[10];
  struct simpul *left;
  struct simpul *right;
  struct simpul *kanan;
};

struct simpul  *kiri= NULL;
struct simpul  *penunjuk;
struct simpul  *kanan = NULL;
```

```
char  nim1[11];
char  nama1[10];

void  insert_kiri(void);
void  insert_tengah(void);
void  insert_kanan(void);
void  delete_kiri(void);
void  delete_tengah(void);
void  delete_kanan(void);
void  tampil(void);
void  pesan1(void);
void  pesan2(void);

main(){
  char pilih;
  ClrBarloop:
  {
  clrscr();
   gotoxy(25,5);  puts("Linier  doubly Linked list");
   gotoxy(25,8);  puts("1.  insert");
   gotoxy(25,10); puts("2.  delete");
   gotoxy(25,12); puts("3.  tampilan");
   gotoxy(25,14); puts("4.  exit");
   gotoxy(25,17); printf("    Pilih : ");
   scanf("%x",  &pilih);

   switch(pilih)  {
   case  1:  printf("\n    masukkan  data NIM  = ");
             scanf("%s", &nim1);
             printf("\n     masukkan  data NAMA = ");
             scanf("%s", &nama1);
             if  (kiri ==  NULL)
                 {   insert_kiri();  }
             else { penunjuk = kiri;
                    if (atof(nim1) < atof(penunjuk->nim))
                           {insert_kiri();}
                    else
```

```
                { while (penunjuk->right != NULL)
                        {penunjuk = penunjuk->right;}
                  if (atof(nim1) > atof(penunjuk->nim))
                        {insert_kanan();}
                  else  {insert_tengah();}
                  }
          }
        getch();
        break;

case  2:
        printf("\n    masukkan  NIM yang akan di delete  = ");
        scanf("%s", &nim1);
        if ( kiri == NULL)
             {    pesan1();
                  goto ClrBarloop;}
        else
             { penunjuk = kiri;
               if (atof(nim1) == atof(penunjuk->nim)) {delete_kiri();}
               else
                  { penunjuk = kanan;
                       if (atof(nim1) == atof(penunjuk->nim))
                            {delete_kanan();}
                       else
                            {penunjuk = kanan->left;
                             while (atof(nim1) != atof(penunjuk->nim))
                                    { if (penunjuk == kiri )
                                          {pesan2();goto ClrBarloop;}
                                      else {penunjuk = penunjuk->left;}
                                    }
                             delete_tengah();
                             }
                  }
             }
        getch();
        break;
```

```
    case  3: tampil();
             getch();
             break;

    case  4: exit(0);
            break;
   }
  goto ClrBarloop;
 }
}

void insert_kiri(void) {
  struct  simpul  *p;
  p = (struct  simpul *) malloc(sizeof(struct simpul));
  strcpy(p->nim, nim1);
  strcpy(p->nama, nama1);

  if ( kiri != NULL)   {
     p->right = kiri;
     kiri->left = p;
     kiri = p;
     p->left = NULL;
     printf("\n                    sisip kiri");
    }
    else
    {
    p->left = NULL;
    p->right = NULL;
    kiri = p;
    kanan = p;
    printf("\n                    create  file");
    }
}

void insert_tengah(void) {
  struct  simpul  *p, *q;
  p = (struct  simpul *) malloc(sizeof(struct simpul));
```

```
  strcpy(p->nim , nim1);
  strcpy(p->nama, nama1);

  if (kiri !=  NULL)   {
     q = kiri;
     while (atof(q->nim) < atof(nim1))
            { q = q->right;}
     p->right = q;
     p->left = q->left;
     q->left->right = p;
     q->left = p;
     printf("\n                    sisip tengah");
    }
    else
    {
     p->left= NULL;
     p->right = NULL;
     kiri = p;
     kanan = p;
     printf("\n                    create  file");
    }
}

/*  Buat procedure  insert_kanan……..*/

void  tampil(void) {
 struct  simpul  *p;
 if ( kiri != NULL)  {
   p = kiri;
   while ( p != NULL)
   {
    strcpy(nim1 , p->nim);
    strcpy(nama1, p->nama);
    p = p->right;
    printf("\n              nim  = %s     nama  = %s", nim1, nama1);
   }
 }
```

```
}

/*Buat procedure delete_kiri……................*/

void delete_tengah(void) {
 struct  simpul  *p,  *q;
 if (kiri != NULL)  {
   p = kiri;
   while (atof(p->nim)  != atof(nim1))
          { p = p->right;}
   p->left->right  =  p->right;
   p->right->left  =  p->left;
   strcpy(nama1, p->nama);
   printf("\n            nim =  %s    nama = %s  ", nim1, nama1);
   free(p);
 }
  else  printf("\n           list kosong");
}

void delete_kanan(void) {
 struct  simpul  *p;
 if (kanan != NULL)  {
   p = kanan;
   kanan = p->left;
   kanan->right = NULL;
   strcpy(nim1 , p->nim);
   strcpy(nama1, p->nama);
   printf("\n            nim =  %s   nama = %s  ", nim1, nama1);
   free(p);
 }
  else  printf("\n          list kosong");
}

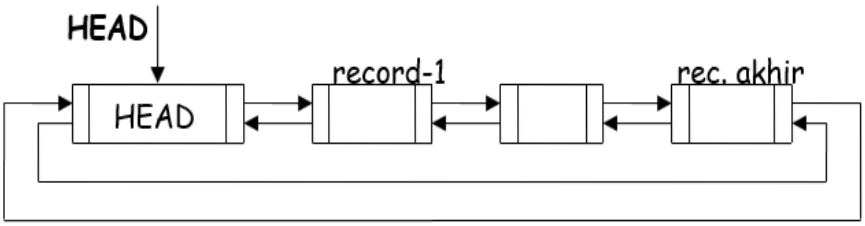
 void pesan1(void) {
   gotoxy(25,22);
   printf("list kosong");
   getch();
```

```
 }

void pesan2(void) {
   gotoxy(25,22);
   printf("\n                    nim tidak ada di list");
   getch();
 }
```

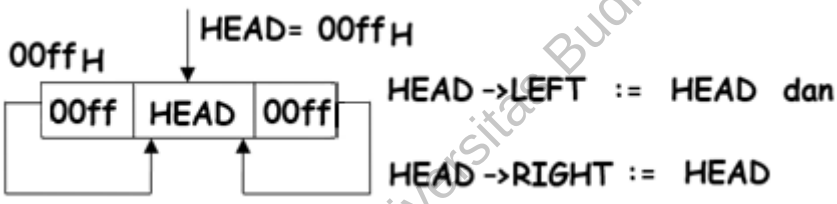
Latihan: Buat procedure insert_ kanan dan delete_kiri

Circular Doubly Linked List

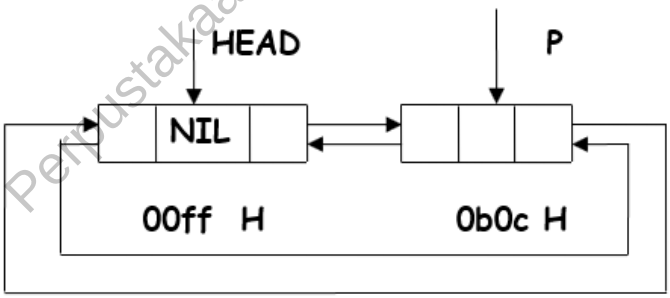


Seperti pada circular singly linked list maka pada circular doubly linked list juga diperlukan elemen HEAD yaitu suatu elemen kosong .

List dalam keadaan kosong.



List berisi 1 elemen

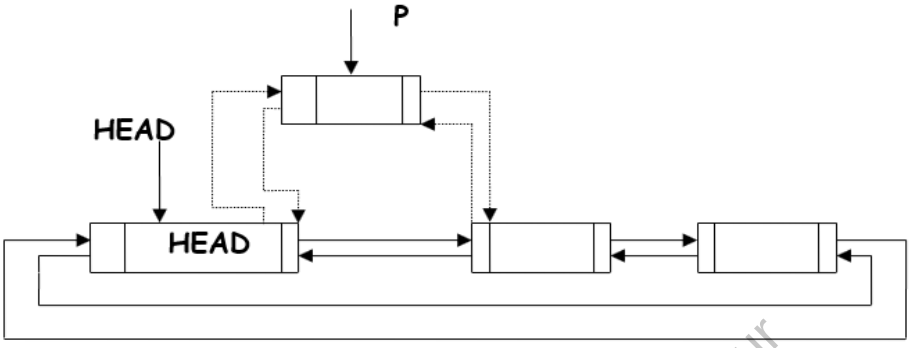


Pada saat list > 1 elemen:

Record-1 berada pada posisi HEAD->RIGHT
sedang record akhir pada posisi HEAD->LEFT

1. Penambahan

Sisip Awal



Untuk procedure yang lain lihat langkah-langkah circular singly linked list.

```
struct simpul
        { char  nim[11];
          char nama[10];
          struct simpul *left;
          struct simpul *right; };
```

2. Aplikasi program circular doubly linked list dalam bahasa pemrograman C :

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

struct  simpul {
  char nim[11];
  char nama[10];
  struct simpul *left;
  struct simpul *right;
};

struct  simpul  *head = NULL;
struct  simpul  *penunjuk;
```

```
char  nim1[11];
char nama1[10];
void  init(void);
void  insert_awal(void);
void  insert_tengah(void);
void  insert_akhir(void);
void  delete_awal(void);
void  delete_tengah(void);
void  delete_akhir();
void  tampil(void);
void  pesan1(void);
void  pesan2(void);

main() {
  char pilih;
  init();
  ClrBarloop:
  {
    clrscr();
    gotoxy(25,5);  puts("Circular  doubly  Linked list");
    gotoxy(25,8);  puts("1.  insert");
    gotoxy(25,10); puts("2.  delete");
    gotoxy(25,12); puts("3.  tampilan");
    gotoxy(25,14); puts("4.  exit");
    gotoxy(25,17); printf("    Pilih : ");
    scanf("%x",  &pilih);

   switch(pilih)  {
   case  1:
            printf("\n            masukkan  data NIM  = ");
            scanf("%s", &nim1);
            printf("\n            masukkan  data NAMA = ") ;
            scanf("%s", &nama1);
            if  (head->right ==  head)
                    {    insert_awal();   }
            else { penunjuk = head->right;
                     if (atof(nim1) < atof(penunjuk->nim))
                            {   insert_awal();  }
                     else
                     { while (penunjuk->right != head)
```

```
                        {penunjuk = penunjuk->right;}

                if (atof(nim1) > atof(penunjuk->nim))
                        {   insert_akhir(); }
                else  {   insert_tengah(); }
                }
            }
        getch();
        break;

case  2:
        printf("\n         masukkan  NIM yang akan di delete  = ");
        scanf("%s", &nim1);
        if ( head->right == head)
                { pesan1();
                  goto ClrBarloop; }
        else
            { penunjuk = head->right;
              if (atof(nim1) == atof(penunjuk->nim))
                      {delete_awal();}
              else
                  { penunjuk = head->left;
                      if (atof(nim1) == atof(penunjuk->nim))
                              {delete_akhir();}
                      else
                          {penunjuk = penunjuk->left;
                          while (atof(nim1) != atof(penunjuk->nim))
                                  { if (penunjuk == head )
                                          {pesan2();goto ClrBarloop;}
                                    Else
                                          {penunjuk = penunjuk->left;}
                                  }
                          delete_tengah();
                          }
                  }
            }
        getch();
        break;

case  3: tampil();
        getch();
```

```
            break;
   case  4: exit(0);
            break;
   }
  goto ClrBarloop;
 }
}

void  init(void) {
  struct  simpul  *p;
  p = (struct simpul *) malloc(sizeof(struct simpul));

  strcpy(p->nim   , 00000);
  strcpy(p->nama , "head");
  head  = p;
  p->left = p;
  p->right = p;
 }

void  insert_awal(void) {
   struct  simpul  *p;
   p = (struct  simpul *) malloc(sizeof(struct simpul));
   strcpy(p->nim , nim1);
   strcpy(p->nama, nama1);

   if (head->left  !=  head)  {
      p->right = head->right;
      p->left = head;
      head->right->left = p;
      head->right = p;
      printf("\n                        sisip awal");
     }
     else
     {
       head->right =  p ;
       head->left = p;
       p->right =  head ;
       p->left = head;
       printf("\n                        create  file");
     }
 }
```

```
void insert_tengah(void) {
    struct  simpul  *p,  *q;
    p = (struct  simpul *) malloc(sizeof(struct simpul));
    strcpy(p->nim , nim1);
    strcpy(p->nama, nama1);

    if (head->left  !=  head)   {
     q = head->right;

     while (atof(q->nim) < atof(nim1))
            { q = q->right;}
     p->right = q;
     p->left = q->left;
     q->left->right = p;
     q->left = p;
     printf("\n                    sisip tengah");
    }
    else
    {
     head->right = p;
     head->left = p;
     p->left = head ;
     p->right = head;
     printf("\n                    create  file");
    }
 }

void insert_akhir(void) {
   struct  simpul  *p,  *q;
   p = (struct  simpul *) malloc(sizeof(struct simpul));
   strcpy(p->nim , nim1);
   strcpy(p->nama,nama1);
   if (head->right  == head)   {
       head->right = p;
      head->left = p;
      p->left = head;
      p->right = head;
      printf("\n                    create  file");
    }
    else
```

```
    {
        p->left = head->left;
        p->right = head;
        head->left->right = p;
        head->left = p;
        printf("\n                    sisip akhir");
    }
}

void tampil(void) {
 struct  simpul  *p;
 if ( head->left != head)  {
   p = head->right;
   while ( p  != head )    {
    strcpy(nim1 , p->nim);
    strcpy(nama1, p->nama);
    p = p->right;
    printf("\n            nim  =  %s  nama1 = %s", nim1, nama1);
   }
 }
}

void delete_awal(void)  {
 struct  simpul  *p;
 if (head->left  !=  head)  {
   p = head->right;
   head->right  = p->right;
   strcpy(nim1 , p->nim);
   strcpy(nama1, p->nama);
   printf("\n              nim =  %s  nama = %s  ", nim1, nama1);
   free(p);
 }
  else  printf("\n                    list kosong");
}

 void delete_tengah(void) {
  struct  simpul  *p,  *q;
 if (head->left != head)
 { p = head->right;
   while (atof(p->nim)  !=  atof(nim1))
          {  p = p->right; }
```

```
   p->left->right  =  p->right;
   p->right->left  =  p->left;
   printf("\n                nim =  %s  nama = %s  ", nim1, nama1);
   free(p);
 }
  else  printf("\n                    list kosong");
}

void delete_akhir(void)  {
 struct  simpul  *p,  *q;

 if (head->right != head)  {
   p = head->left;
   p->left->right = head;
   head->left = p->left;
   strcpy(nim1 , p->nim);
   strcpy(nama1, p->nama);
   printf("\n              nim =  %s   nama = %s  ", nim1, nama1);
   free(p);
 }
  else  printf("\n                     list kosong");
}

 void pesan1(void) {
   gotoxy(25,22);
   printf("list kosong");
   getch();
 }

 void pesan2(void) {
   gotoxy(25,22);
   printf("\n                        nim tidak ada di list");
   getch();
}
```

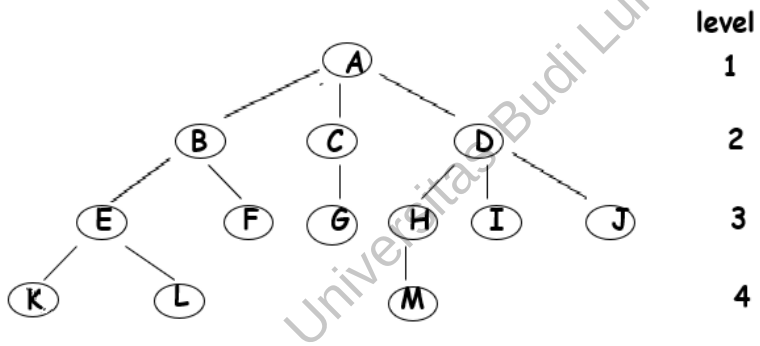
Struktur Data Non Linier

Terdiri dari
- Struktur pohon
- Graph

1. Struktur Pohon (Tree Structure)

a. Definisi Pohon

susunan dari satu atau lebih simpul (node) yang terdiri dari satu simpul khusus yang disebut akar (root) sedang sisanya membentuk subtree dari akar



Akar dari struktur pohon ini adalah A

Satu simpul akan berisi :
- Informasi (misal: A, B dst.)
- cabang-cabang (link) yang menghubungkan ke simpul-simpul yang lain

Simpul A sebagai akar mempunyai 3 link yang membentuk **Subtree** B, C dan D.
Jumlah subtree dari satu simpul disebut: **Derajat (Degree)**
Derajat dari simpul :

A = 3
B = 2
C = 1
G = 0

Simpul yang mempunyai derajat = 0 disebut:

SIMPUL TERMINAL atau DAUN (LEAF)

contoh :
simpul daun dalam gambar diatas adalah
K, L, F, G, M, I dan **J**.

Struktur pohon yang terkenal adalah struktur geneologi (silsilah). Dalam struktur silsilah dikenal adanya simpul anak (children) dan orangtua dari anak (parent).

Contoh: **anak (child)** dari **D** adalah **H, I** dan **J**.
orangtua (parent) dari **D** adalah **A**

Anak dari orangtua yang sama disebut **SIBLING (mis: H, I, J)**

b. Derajat (degree) suatu pohon
adalah derajat maksimum dari suatu simpul dalam pohon

contoh: derajat pohon dari gambar diatas adalah 3.

Nenek moyang dari suatu simpul adalah seluruh simpul-simpul yang ada sepanjang lintasan dari akar sampai simpul tersebut.

Contoh: nenek moyang M adalah A, D dan H

c. Kedalaman (HEIGHT atau DEPTH)
Dari suatu pohon ditentukan oleh level maksimum dari simpul dalam pohon.

Contoh: kedalaman pohon dari gambar diatas adalah 4

HUTAN (FOREST)

adalah susunan dari beberapa pohon

Bila akar A dihilangkan maka akan diperoleh hutan dengan 3 pohon yaitu:

- B (E (K,L) , F)
- C (G)
- D (H (M) , I, J)

Ada 2 cara untuk menyatakan struktur pohon, yaitu dengan:

1. gambar
2. daftar (list)

Pernyataan dengan gambar adalah seperti terlihat pada gambar diatas, sedang kalau dengan list adalah sebagai berikut :

(A (B (E (K,L), F), C (G), D(H (M), I, J)))

Proses dalam struktur data non linier, bentuk pohon akan lebih mudah digambarkan bila diketahui :

- **n (jumlah simpul atau node)**
- **k (derajat pohon)**

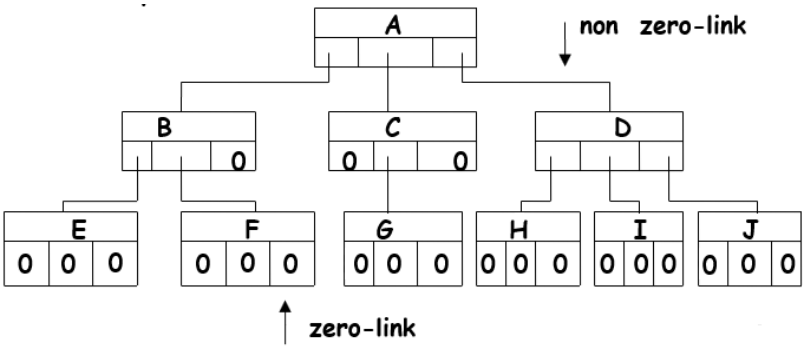
Dari data n dan k maka dapat dihitung :

Jumlah Link = n . k
Jumlah Null-Link = n (k - 1) + 1
Jumlah Non Zero-Link = n - 1

d. Struktur node k-ary

D A T A		
child-1	.....	child-k

Pohon 3-ary :



Dari gambar diatas diketahui: **n = 10 dan k = 3**
maka dapat dihitung:

jumlah link = n.k = 3.10 = 30
jumlah null-link = n(k-1) + 1
= 10 (3-1) 1
= 21
jumlah non zero-link = n - 1 = 10 - 1
= 9

Bila

k = 2 maka pohon disebut POHON BINER.

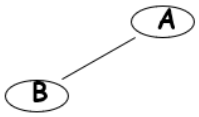
2. POHON BINER (BINARY TREE)

Adalah tipe yang sangat penting dari struktur data.

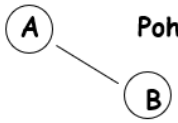
Dalam struktur pohon biner hanya dikenal **Subtree Kiri** dan **Subtree Kanan** saja.

Simpul dalam pohon biner adalah:

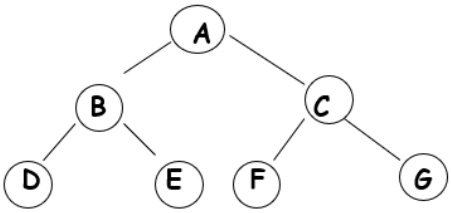
susunan dari simpul-simpul yang masing-masing bisa kosong atau terdiri dari akar dan dua pohon biner yang terpisah dan disebut subtree kiri dan subtree kanan.



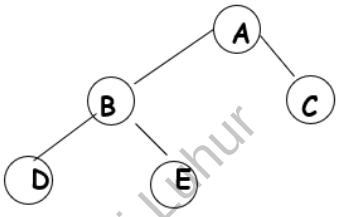
Pohon biner dengan subtree kanan kosong.



Pohon biner dengan subtree kiri kosong



Full Binary Tree



Complete Binary

3. Tree

Pohon Biner Penuh ("Full Binary Tree")

adalah pohon biner yang mempunyai simpul atau node lengkap dari level 1 sampai dengan i

Pohon Biner Lengkap ("Complete Binary Tree")

adalah pohon biner yang mempunyai simpul dengan nomor urut dari 1 sampai dengan n

Jumlah maksimum simpul dari pohon biner dengan level i adalah:

$2^i - 1$ dimana $i >= 1$

Jumlah simpul pada level i adalah :

2^{i-1}

contoh:

Bila kedalaman suatu pohon = 4 maka jumlah simpul dari pohon tersebut:

$$2^4 - 1 = 15$$

Banyaknya simpul pada:

level 1 : 2^{1-1} = 1
level 2 : 2^{2-1} = 2
level 3 : 2^{3-1} = 4
level 4 : 2^{4-1} = 8

Kedalaman **minimal** dari pohon biner adalah :

($\log_2$ n + 1) dimana n = jumlah simpul

contoh: bila n = 15

maka kedalaman minimal = $\log_2 15 + 1$

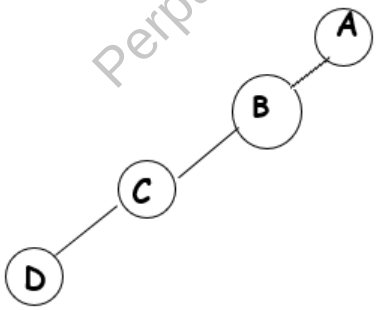
$$= \frac{\log 15}{\log 2} + 1$$

$$= 1{,}17 / 0.3 + 1$$

$$= 3 + 1 = 4$$

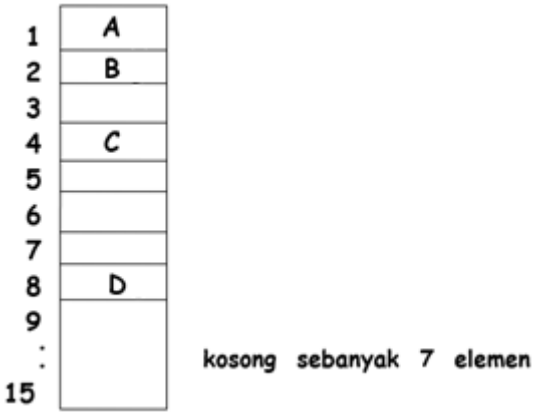
a. Skewed tree

adalah pohon biner yang miring ke kiri atau ke kanan, atau dengan kata lain pohon biner dengan subtree kiri kosong atau kanan kosong.



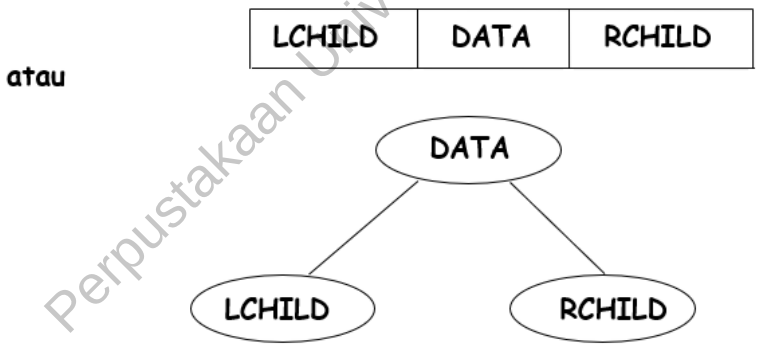
Bentuk pohon seperti ini bila disim pan dalam bentuk array sangat memboroskan.

Representasinya adalah sebagai berikut:



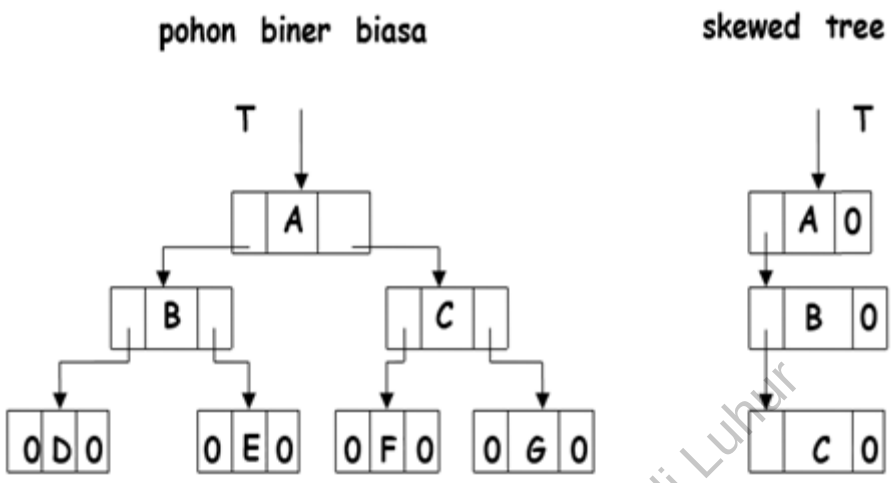
Terlihat dari contoh diatas bahwa penyimpanan data dalam memory dari pohon biner hanya menguntungkan kalau pohon binernya penuh sehingga tidak memboroskan tempat.

Untuk menanggulangi ini maka perlu menggunakan representasi **linked list**, dimana masing-masing simpul akan mempunyai 3 field yaitu:




```
struct  nama_simpul
        {
          tipe  data  DATA ;
          struct   nama_simpul *RCHILD,  *LCHILD ;
         };
```

b. Representasi link



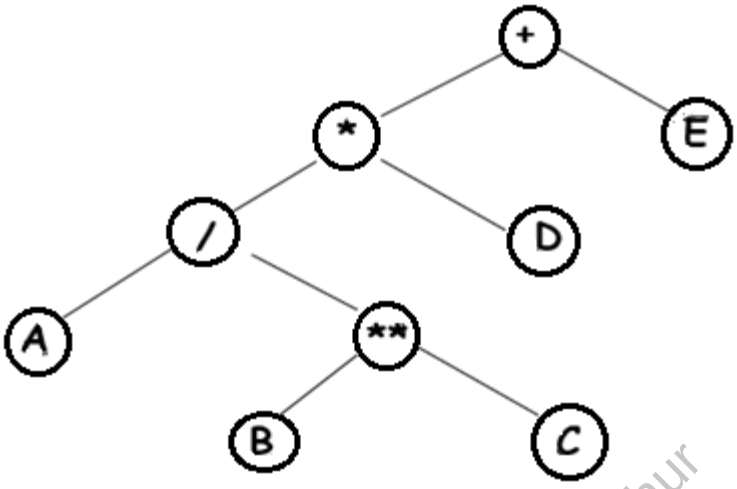
4. Penelusuran pohon biner (Binary Tree Traversal)

Adalah ide untuk melakukan penelusuran (traversing) atau kunjungan (visiting) masing-masing simpul sebanyak satu kali.
Penelusuran satu pohon biner penuh akan menghasilkan urutan linier dari informasi dalam pohon itu.

Ada 3 macam traversal:
- Inorder Traversal
- Preorder Traversal
- Postorder Traversal

a. Inorder traversal

Dimulai dari simpul paling kiri bawah kemudian ke akar dari simpul tersebut dilanjutkan ke kanan dari akar yang ditinjau, kemudian cari simpul paling kiri dari subtree kanan tersebut, begitu seterusnya sampai tidak bisa berpindah ke kanan.



Dari contoh diperoleh persamaan Inorder :

A / B ** C * D + E

Langkah-langkah:

1. Telusuri subtree kiri dalam inorder
2. Proses simpul akar
3. Telusuri subtree kanan dalam inorder

Prosedure INORDER menggunakan stack:

```
void   RINORDER(void) {
    struct  simpul  *p;
     if  (T  != NIL)
       {  P = T;
         TOP = 0;
          do   {
               while  (  P  !=  NULL)
                    {  PUSH();
                       P  =  P->LCHILD;}
              P = POP(S, TOP);
              printf ( "%i ", DATA(P));
              P  = P->RCHILD; }
          while  ( P== NULL) &&  (STACK == EMPTY)  }
   else
```

```
        { printf("POHON KOSONG");
          return(0);  }
}
```

Procedure INORDER menggunakan recursive:

```
void  RINORDER( void){
  if  (T != NULL)
        { if (T->LCHILD != NULL)
              { T = T->LCHILD;
                RINORDER();}
          printf("%i ", T->DATA);
          if (T->RCHILD  != NULL)
              { T =  T->RCHILD;
                RINORDER();}
        }
    else
        {   printf(" EMPTY  TREE");
            return(0);}
}
```

Terlihat dari contoh diatas, bahwa penulisan dengan recursive lebih sederhana dibanding dengan pemrograman biasa.

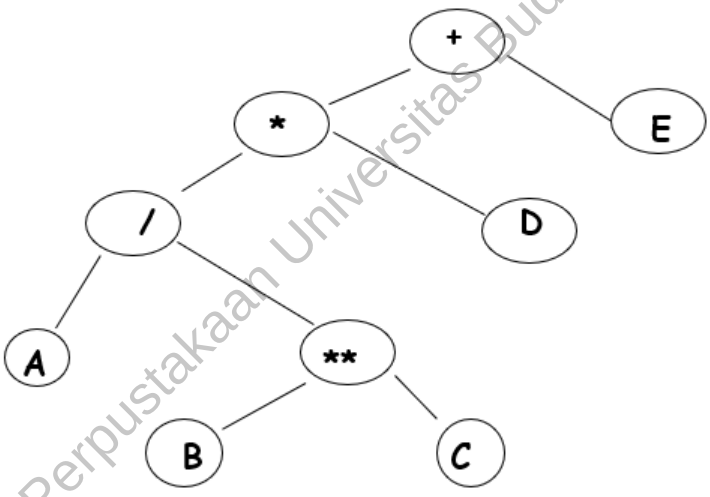
Pelacakan dengan rekursive

IF T <> NIL THEN

- **- ya → - ya → - ya → - tidak**
  - print 'A'
  - tidak
  - print '/'
  - ya → - ya → - tidak
  - print 'B'
  - tidak
  - print ''**
  - ya → - tidak

- print'C'
- tidak
- print '*'
- ya → - tidak
  - print 'D'
  - tidak
- print '+'
- ya → - tidak
  - print 'E'
  - tidak

b. Preorder traversal



Langkah-langkah:

1. Proses simpul akar
2. Telusuri subtree kiri dalam preorder
3. Telusuri subtree kanan dalam preorder

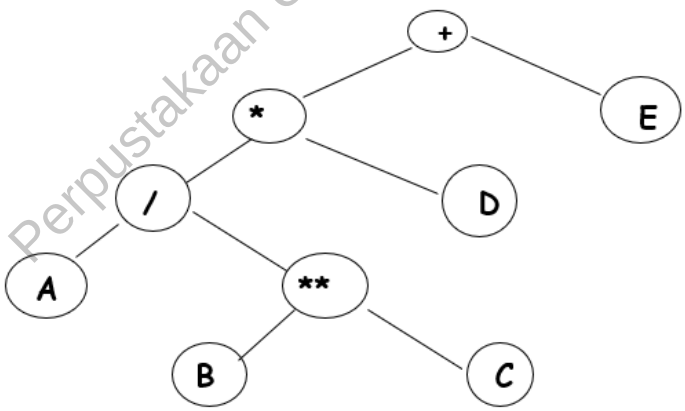
Persamaan preorder adalah:

+ * / A ** B C D E

Prosedure PREORDER

```
void   RPREORDER(void)
{  if  ( T != NULL)
       {  printf(" %i  ",  T->DATA);
          if  ( T->LCHILD != NULL)
             { T = T->LCHILD;
               RPREORDER();
             }
          if  ( T->RCHILD != NULL)
             {  T =  T->RCHILD;
                RPREORDER(); }
              }
   else
      {  printf(" EMPTY TREE");
       return(0);}
}
```

c. Postorder traversal



Langkah-langkah:

1. Telusuri subtree kiri dalam postorder
2. Telusuri subtree kanan dalam postorder
3. Proses simpul akar

Persamaan postorder adalah:

A B C ** / D * E +

Procedure rekursive dari postorder adalah sebagai berikut:

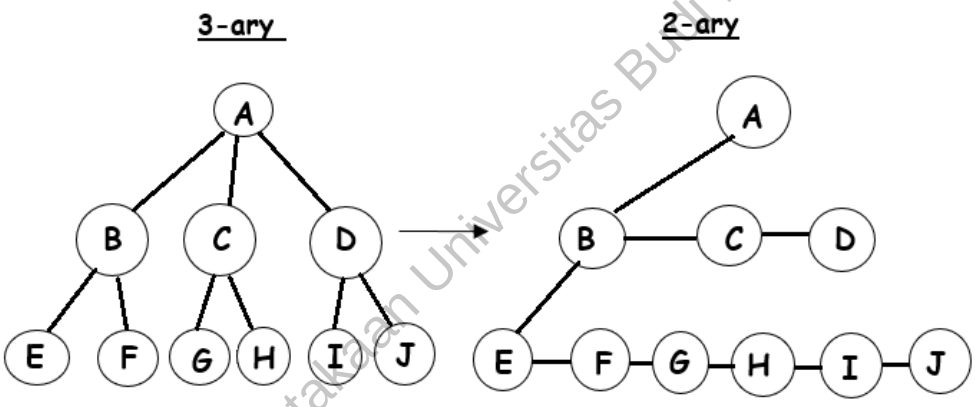
```
void   RPOSTORDER(void)
{ if  ( T != NIL )
     {  if  (T->LCHILD  != NULL)
           {  T = T->LCHILD;
              RPOSTORDER(); }
        if  ( T->RCHILD != NULL)
           { T = T->RCHILD;
             RPOSTORDER(); }
        printf(" %i  ", T->DATA);
     }
  else
   {  printf(" EMPTY TREE ");
      return(0);
   }
}
```

Representasi dari dari pohon k-ary ke binary

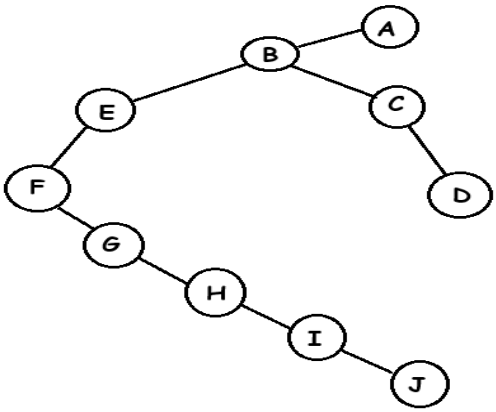
Representasi dari pohon k-ary ke binary mempunyai struktur simpul sebagai berikut:

DATA	
CHILD	SIBLING

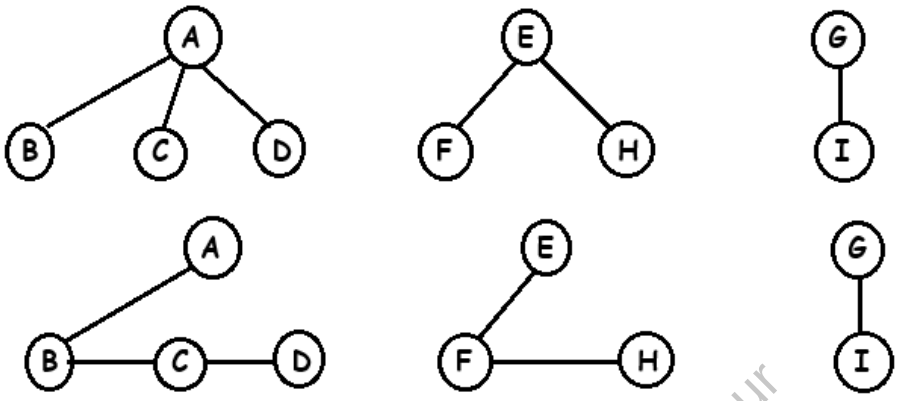
```
struct nama_simpul
    { tipe data    DATA;
      struct nama_simpul   *CHILD, *SIBLING;}
```



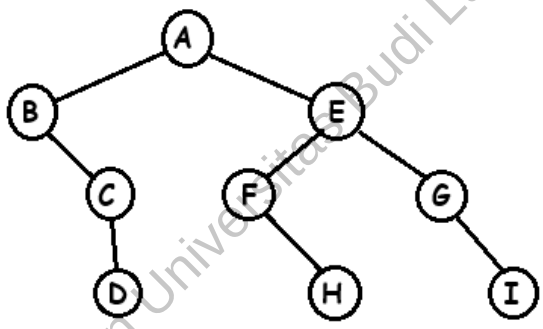
Atau :



Kalau ada 3 pohon, kemudian disatukan menjadi satu pohon



atau



1. Pohon biner berbenang (threaded binary tree)

Struktur pohon biner berbenang dapat digunakan untuk memanfaatkan "null-link" yang ternyata cukup banyak jumlahnya dalam struktur pohon biner, yaitu :

jumlah null-link biner = n(k-1) + 1 = n(2-1) + 1 = n + 1

Idenya yaitu menggantikan null-link dengan link yang disebut benang(thread), ke simpul yang lain dalam pohon.

Struktur simpulnya adalah sebagai berikut:

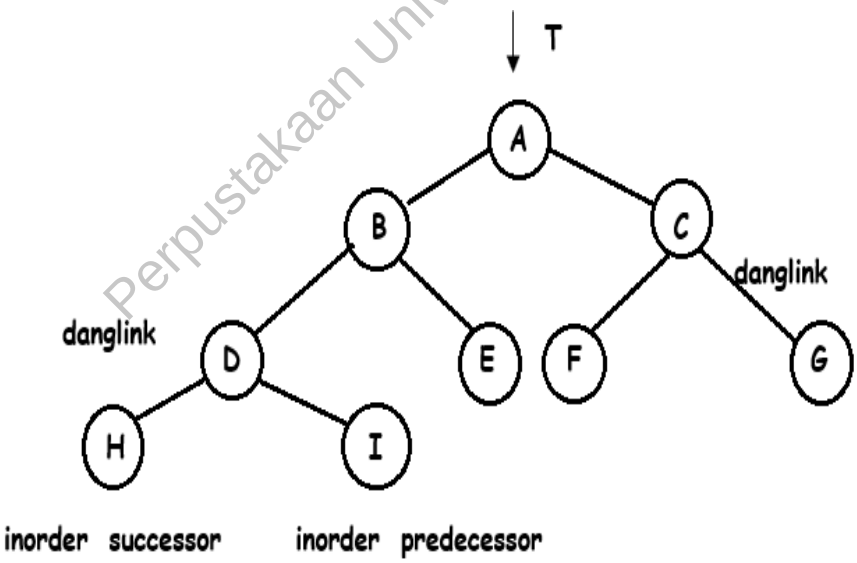
LBIT	LCHILD	DATA	RCHILD	RBIT

LBIT, RBIT = 1;
bila LCHILD atau RCHILD merupakan link biasa

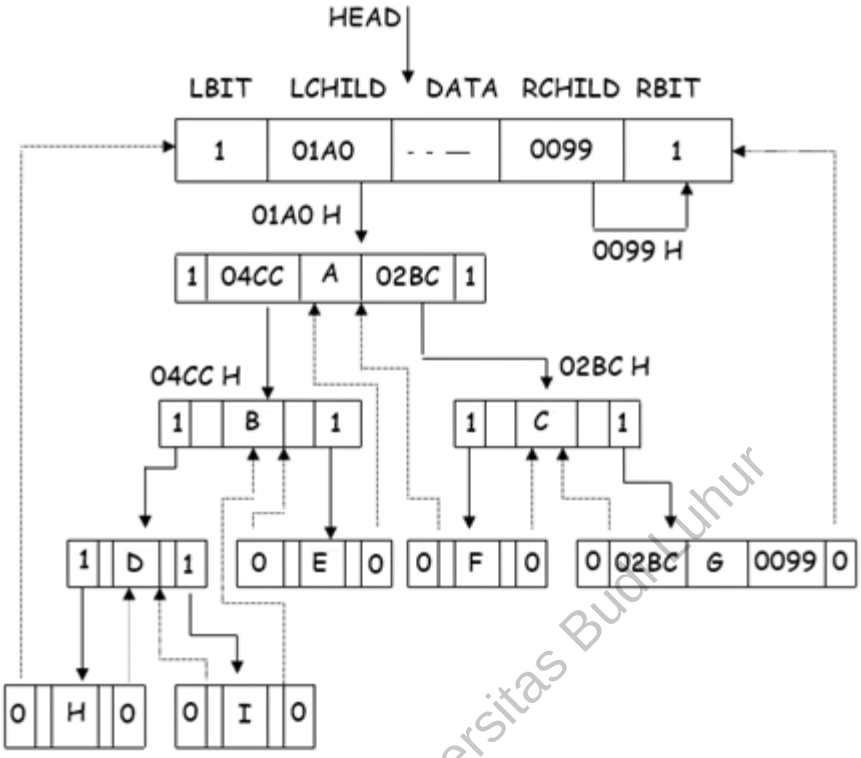
LBIT, RBIT = 0;
bila LCHILD atau RCHILD merupakan link benang

Agar supaya 2 link benang yang tidak menunjuk kemana-mana **(Danglink)** yaitu link paling kiri dari subtree kiri dan link paling kanan dari subtree kanan dapat diisi dengan link benang maka dibuat suatu simpul kepala.

2. Pohon biner berbenang tanpa simpul head



3. Pohon biner berbenang dengan simpul head



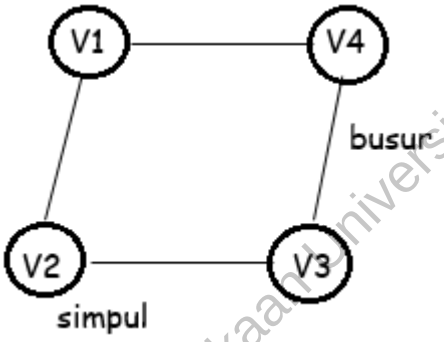
Graph

Dinyatakan dalam suatu fungsi G = (V , E) dimana
V = kumpulan simpul/ vertex/ node
E = kumpulan busur / edge/ arc

Tiap busur pasti menghubungkan 2 simpul
Ada 2 macam graph

1. Graph tak terarah (Undirected graph)
2. Graph terarah (Directed graph)

1. Graph tak terarah (undirected graph)



V1-V2 = V2-V1

V1-V4 = V4-V1

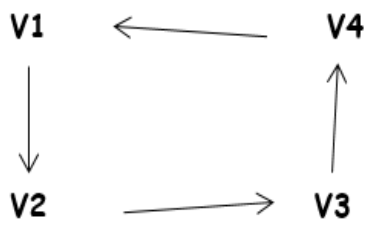
dst

Graph tak terarah adalah graph yang menghubungkan 2 simpul V1 ke V2 dan V2 ke V1 (2 arah)

Bila simpul = n maka graph tak terarah komplit akan mempunyai busur sama dengan

$$n (n - 1) / 2$$

2. Graph terarah (directed graph)



Graph terarah adalah graph yang hanya dapat menghubung kan V1 ke V2 saja (1 arah)
(V1-V2) <> (V2-V1)

Maksimum jumlah busur dari n simpul adalah :

$$n \ (\ n - 1 \ )$$

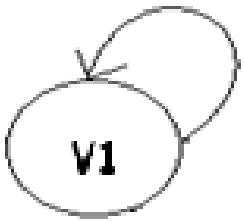
3. Multigraph

Multi graph adalah 2 buah simpul V1-V2 yang dihubungkan oleh busur busur yang jaraknya berbeda-beda

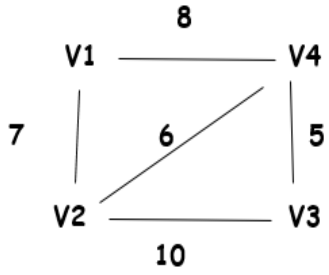
4. CYCLE

Definisi

Cycle adalah suatu lintasan yang kembali ke simpul semula



Bobot (panjang) lintasan



Bobot (panjang) lintasan adalah jumlah busur yang menghubungkan simpul Vx ke simpul Vy

panjang dari V1 ke V4 adalah :

busur 1, 2, 4 ⟶ 7 + 6 = 13

busur 1, 2, 3,4 ⟶ 7 + 10 + 5 = 22 (lintasan kritis)

busur 1, 4 ⟶ 8

GRAPH CONECTED

Definisi:

Suatu graph adalah connected jika untuk setiap pasangan simpul Vi dan Vj ada lintasan yang menghubungkannya.

GRAPH UNCONECTED

Definisi

Suatu graph non connected berisi paling sedikit 1 pasangan yang tidak terhubung

Representasi Graph

Terdiri dari:

1. Adjacency matrix
2. Adjacency List
3. Adjaency Array

1. Adjacency matrix

Orientasinya ke simpul dari graph

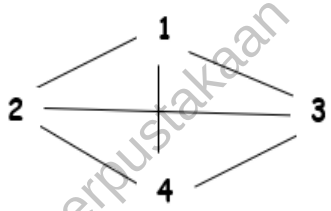
Ukuran : **A[n x n]**

dimana A = matrix
n = jumlah simpul

Definisi :

a_i = 1 ; jika ada busur yang menghubungkan simpul i dan j
= 0 ; jika tidak ada hubungan

Undirected



$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Δ atas & Δ bawah simetri

Directed

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

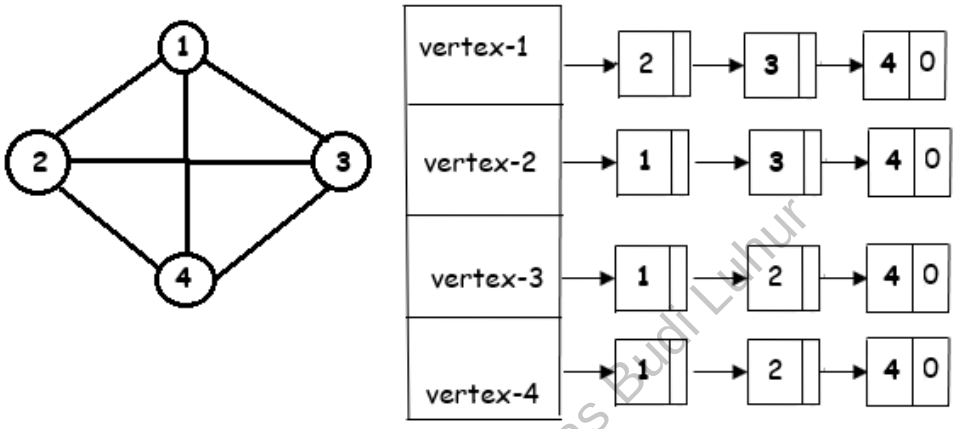
kolom menyatakan **IN - DEGREE**
baris menyatakan **OUT - DEGREE**

2. **Adjacency list**

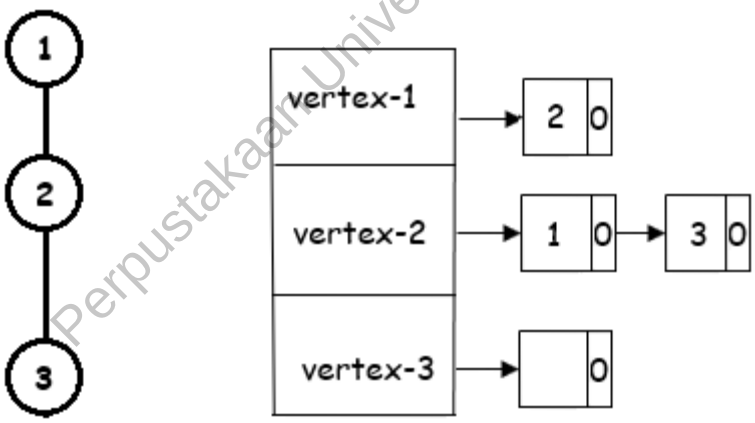
Representasinya dalam bentuk list berangkai

Keuntungan: bisa dilakukan penambahan simpul dengan mudah, karena sifatnya dinamis

Undirected Graph



Directec Graph

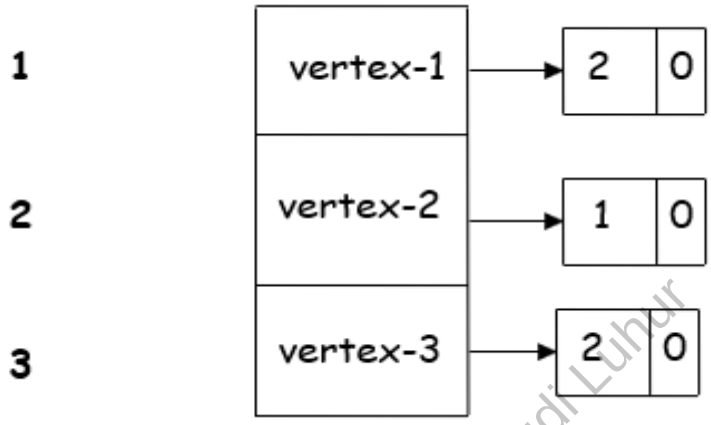


Derajat dari tiap simpul dalam undirected graph bisa ditentukan dengan menghitung jumlah simpul dalam adjacency list.

Sedangkan dalam directed graph derajat keluaran (**outdegree**) dapat ditentukan dari jumlah node yang terhubung pada adjacency listnya, tapi mengalami kesulitan dalam melihat derajat masukan (**indegree**) dari tiap simpul.

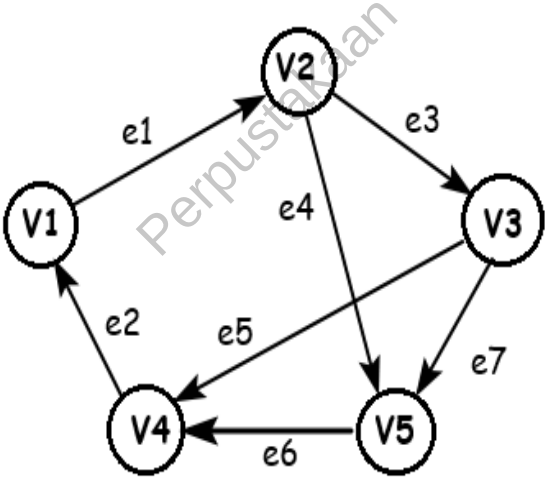
Untuk itu disajikan bentuk Inverse Adjacency List yang memperlihatkan jumlah derajat masukan dari tiap simpul.

INVERSE ADJACENCY LIST



3. Adjacency Array

Lebih cocok untuk graph berarah, untuk setiap e_k mempunyai pasangan simpul asal Vi dan simpul tujuan Vj.



	Vasal	Vtujuan
e1	1	2
e2	4	1
e3	2	3
e4	2	5
e5	3	4
e6	5	4
e7	3	5

Asal	1	4	2	2	3	5	3
Tujuan	2	1	3	5	4	4	5

Gabungan dari kedua array ini menjadi 1 array adalah sebagai berikut:

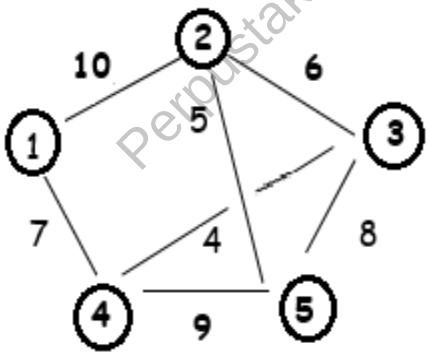
3	5	3	2	2	4	1	X	2	1	3	5	4	4	5

simpul asal simpul tujuan

4. Aplikasi GRAPH

a. Critical Path (lintasan kritis)
b. Minimum Spanning Tree (MST): Bagaimana mencapai semua simpul suatu simpul asal, dengan total lintasan minimum

Lintasan Kritis: menggunakan Graph berbobot



Graph G

Cari lintasan kritisnya
Simpul asal : 1
Simpul tujuan : 5

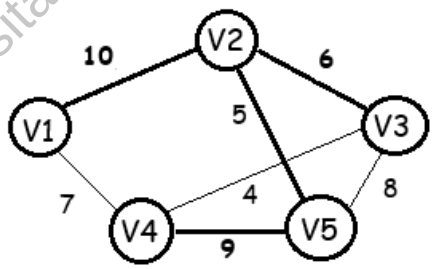
	lintasan					bobot
alternatif :	1	2	5			15
	1	2	3	5		24
	1	2	3	4	5	29
	1	4	5			16
	1	4	3	5		19
	1	4	3	2	5	22

diperoleh: lintasan kritis = 29
lintasan terpendek = 15

<u>Minimum Spanning Tree</u>

Lihat gambar Graph *G*

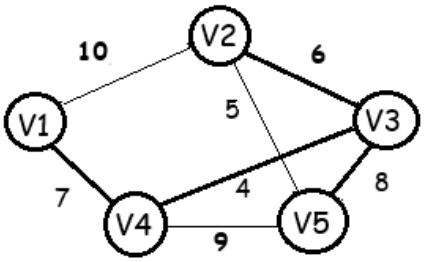
G' = <u>subgraph dari *G*</u>



Bobot = 10 + 6 + 5 + 9
= 30

G" = <u>subgraph dari *G*</u>

membentuk struktur pohon
spanning tree dari *G*

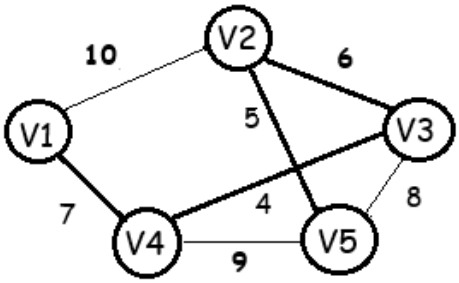


Bobot = 7 + 4 + 6 + 8 = 25

G''' = <u>subgraph dari *G*</u>

Bobot = 7 + 4 + 6 + 5

Maka MST = 22



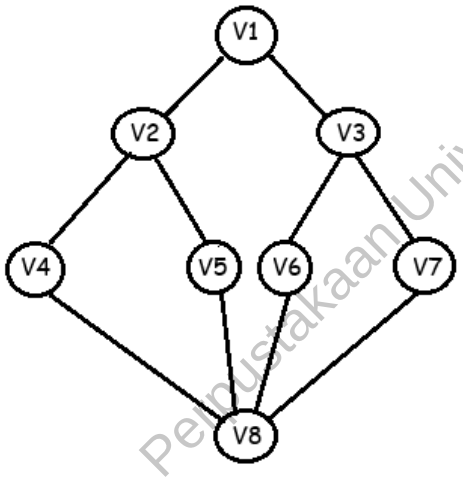
Penelusuran / Pencarian Graph

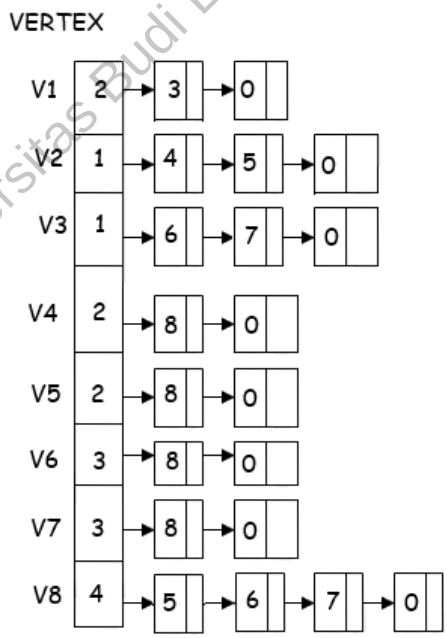
Dapat dilakukan dengan 2 cara yaitu :

- Depth First Search (DFS)
- Breadth First Search (BFS)

1. Depth First Search (DFS)

Penelusuran dengan DFS pada graph tak terarah dengan melakukan pengecekan pada simpul dengan kedalaman pertama dari simpul yang ditinjau





Procedur atau fungsi dalam Bahasa pemrograman C adalah sebagai berikut:

```
void   DFS( V ) {
   /* V adalah node yang ditinjau  *
      visited adalah array untuk node yang sudah dikunjungi
      dengan harga awal seluruhnya sama dengan nol  */
```

```
VISITED[ V ]     ←——   1

for  masing-masing vertex(w) adjacent to v
{    if  ( VISITED[W] ==  0)
            { DFS(W)   }
}
```

Dari gambar diatas akan diperoleh urutan :

V1 → V2 → V4 → V8 → V5, V6 → V3 → V7

2. Breadth First Search (BFS)

Berbeda dengan cara DFS, dengan BFS penelusuran akan diawali dari node-1 kemudian melebar pada adjacent node dari node-1 dan diteruskan pada node-2, node-3 dst.

Procedur atau fungsi dalam Bahasa pemrograman C adalah sebagai berikut:

```
Void BFS( V );
{
   /* Q adalah antrian  dari  vertex  V, sedang  VISITED  adalah  array
      dari  vertex yang sudah ditinjau dengan  harga  awal = 0 dan bila
      sudah  ditinjau  =  1
      Bila akan  ditinjau  vertex  dimasukkan  dalam  antrian,  sedang bila
      sudah  ditinjau  vertex  akan  di delete  */

            VISITED[ V ]   ←—  1
            while   ( Q  not  empty )
            {
                DELETE  Q(V, Q)
                for  (all  vertices  W  adjacent  to  V  )
                    {
                        IF  ( VISITED (W) = 0 ) {
```

```
                    ADDQ(W,Q)
                    VISITED ←—— 1 }
                }
            }
}
```

Dengan procedure BFS akan diperoleh urutan sebagai berikut:

V1 ⟶ V2, V3 ⟶ V4, V5, V6, V7 ⟶ V8

Search (Pencarian)

File adalah kumpulan record-record sejenis dimana masing-masing record mempunyai lebih dari satu field. Field-field tersebut diperlukan untuk memberikan suatu identifikasi bagi masing-masing record. Key(kunci) field untuk identifikasi record akan tergantung pada aplikasi tertentu.

Misal: File mahasiswa menggunakan field yang berisi NOMOR INDUK MAHASISWA (NIM)

Ada beberapa cara searching, yaitu:

a. linear search (sequential search)
b. binary search
c. Fibonacci search

Dalam melakukan search (pencarian) ini diasumsikan file dalam keadaan sudah sort (urut).

1. Linear Search

Pencarian dimulai dari record-1, diteruskan ke record berikutnya yaitu record ke-2, ke-3 ... dst, sampai diperoleh isi record sama dengan informasi yang dicari. Oleh karena itu file pada Linear search tidak mutlak harus urut

Bila N = jumlah record dalam file
X = informasi yang akan dicari dalam file
K = nama file
K[N+1] = X digunakan sebagai sentinel elemen

maka procedurnya adalah sebagai berikut :

```
#include <stdio.h >
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
```

```
#include <dos.h>
#define   n     20

void linier(void);
void binary(void);
void fibonaci(void);

static int K[n+1] = { 0, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,
                              65, 70, 75, 80, 85, 90, 95, 100, 105, 0,};
int nim;
struct time waktu;
main() {
   int pilih;
   ClrBarloop:
   clrscr();
     printf("\n\n");
     gotoxy(35,5);puts("Coba Search");
     gotoxy(25,7);puts("1. Linier search");
     gotoxy(25,9);puts("2. Binary search");
     gotoxy(25,11);puts("3. Fibonaci search");
     gotoxy(25,13);puts("4. exit");
     gotoxy(29,15);puts("Pilih : ");
     gotoxy(37,15);
     scanf("%x", &pilih);

     switch(pilih)   {
    case 1: printf("\n masukkan data nim = ");
              scanf("%d", &nim);
              linier();
              getch();
              break;
     case 2: printf("\n masukkan data nim = ");
              scanf("%d", &nim);
              binary();
              getch();
              break;
     case 3: printf("\n masukkan data nim = ");
              scanf("%d", &nim);
              fibonaci();
              getch();
              break;
```

```
   case 4: exit(0);
           break;
   }
   goto ClrBarloop;
 }

void linier(void) {
  int I;
  gettime(&waktu); printf("\n");
  cprintf("%d:%d:%d",waktu.ti_hour,waktu.ti_min,waktu.ti_sec);
  printf("\n");

  I=1;
  printf("\n iterasi ke- :");
  while( K[I] != nim)
       {  printf(" %d",I);
          I=I+1;  }
  if (I == (n+1) )
       {   printf(" search gagal ");   }
  else
       { printf(" %d",I);
         printf("\n\n iterasi = %d nim = %d ada di posisi = %d", I,nim, I);
       }
  printf("\n");
  gettime(&waktu); printf("\n");
  cprintf("%d:%d:%d",waktu.ti_hour,waktu.ti_min,waktu.ti_sec);
}

void binary(void){
   int low, mid, high, I;
   gettime(&waktu); printf("\n");
   cprintf("%d:%d:%d",waktu.ti_hour,waktu.ti_min,waktu.ti_sec);
   printf("\n");
   low = 1;
   high = n;
   I = 1;
   printf("\niterasi low high mid");
   while (low <= high)
         { mid = (low + high)/2;
           printf ("\n %d %d %d %d",I,low, high,mid);
           getch();
```

```
          if (nim < K[mid])
             {high = mid - 1;
              I = I+1;}
           Else  { if (nim K[mid])
                   { low = mid + 1;
                     I= I+1;}
               else { printf("\n\nnim = %i ada di posisi %i\n",nim, mid);
                      gettime(&waktu);
                      printf("\n");
                      cprintf("%d:%d:%d", waktu.ti_hour,
                                          waktu.ti_min, waktu.ti_sec);
                      printf("\n");
                      return;}
                }
            }
 }

void fibonaci(void)
{ unsigned int fib[25];
  int I, p, q, m, j, t;
  int iterasi = 0;
  gettime(&waktu); printf("\n");
  cprintf("%d:%d:%d",waktu.ti_hour,waktu.ti_min,waktu.ti_sec);
  printf("\n");

   fib[0] = 0;
   fib[1] = 1;
   for( j = 2; j<15 ; j++)
         {  fib[j] = fib[j-1] + fib[j-2]; }
   printf("\n Fibonacci sequence : ");
   for( j = 0; j<15 ; j++)
         {  printf("%i ", fib[j]); }
  getch();
  j = 0;
   while ( fib[j] <= (n + 1))
             {  j= j+1; }
   m = n + 1 - fib[j-1];
   I = fib[j-2];
   p = fib[j-3];
   q = fib[j-4];
```

```
  printf("\n\n m= %i I= %i p= %i q=%i\n", m,I,p,q);
  getch();
  if (nim K[I])
       { I = I + m;}
   while ( I != 0 )
      { iterasi = iterasi + 1;
        printf("\niterasi ke-%d I = %d K[%d] = %d", iterasi,I,I,K[I]);
        if ( nim < K[I] )
           { if (q != 0)
                { I = I-q;
                  t=p;
                  p = q;
                  q = t-q; }
            else
                I = 0;
           }

     else { if (nim < K[I])
              { if (p != 1)
                   { I = I+ q;
                     p=p-q;
                     q = q-p;
                    }
                else
                   I = 1;
              }
           else {gettime(&waktu); printf("\n");
                  cprintf("%d:%d:%d", waktu.ti_hour,
                                      waktu.ti_min,waktu.ti_sec);
                  printf("\n");
                  return;}
                }
      }
}
```

Cara ini sangat jelek karena memerlukan waktu yang lama. Kemungkinan terjelek adalah memerlukan waktu N+1, sehingga waktu yang diperlukan untuk search adalah **O(n).**

Waktu yang diperlukan untuk search secara umum: **f(n) = O(g(n))**

f(n) untuk menyatakan computing time dari beberapa algoritme
O(n) disebut waktu linear
O(log n) disebut waktu logaritmik, lebih cepat dari waktu linear.

2. Binary Search

Merupakan **metoda terbaik dalam search.**
Memulai search dari lokasi tengah (**MIDDL**E). Kemudian berdasarkan isi dari lokasi ditengah tersebut dapat diberikan 3 kemungkinan :

1. jika X < K[M] : informasi yang dicari ada dibagian bawah dari lokasi middle.
2. jika X = K[M] : record tengah tersebut adalah record yang dicari
3. jika X > K[M] : informasi yang dicari ada di bagian atas dari lokasi tengah (middle)

Demikian seterusnya setiap sudah ditentukan lokasi tengah, sampai diperoleh record yang dicari.

Contoh: suatu urutan data dengan cacah data N = 10.

Data tersebut mempunyai nilai sebagai berikut :
record ke- :

1	2	3	4	5	6	7	8	9	10
75	151	203	275	300	489	524	591	647	727

logika pencarian: cari record yang bernilai 275

iterasi	low	high	middle	compare
1	1	10	5	275 < 300
2	1	4	2	275 > 151
3	3	4	3	275 > 203
4	4	4	4	275 = 275

Pencarian hanya memerlukan 4 langkah (iterasi)

Waktu yang diperlukan pada Binary Search ini adalah: O(log n)

Algoritme dalam Bahasa pemrograman C

```
void binary(void)
{  int low, mid, high, I;
   gettime(&waktu); printf("\n");
   cprintf("%d:%d:%d", waktu.ti_hour, waktu.ti_min, waktu.ti_sec);
   printf("\n");
   low = 1;
   high = n;
   I = 1;
   printf("\niterasi low high mid");

   while (low <= high)
         { mid = (low + high)/2;
            printf ("\n %d %d %d %d",I,low, high,mid);
            getch();

            if (nim < K[mid])
              {  high = mid - 1;
                 I = I+1;}
           Else
             {if (nim K[mid])
                    { low = mid + 1;
                      I= I+1;}
               else
                    { printf("\n\nnim = %i ada di posisi %i\n",nim, mid);
                      gettime(&waktu);
                      printf("\n");
                      cprintf("%d:%d:%d", waktu.ti_hour,
                                          waktu.ti_min,waktu.ti_sec);
                      printf("\n");
                      return;}
                 }
            }
}
```

3. **Fibonacci Search**

Menggunakan cara search dengan memecah subfile menurut Fibonacci Sequence, yaitu:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ……..

diperoleh dari harga awal Fo = 0 dan F1 = 1.
Fibonacci sequence: $F_i = F_{i-1} + F_{i-2}$; $i >= 2$

Keuntungannya:
Dalam prosesnya hanya menggunakan operator pertambahan dan pengurangan saja.
Oleh karena itu waktu rata-rata search lebih cepat dari Binary Search yang menggunakan pembagian dalam proses search-nya.

Contoh : File data sudah dalam kondisi urut

	11	23	36	42	58	65	74	87	94	99
record :	1	2	3	4	5	6	7	8	9	10

carilah record yang berharga 65.

Diketahui :

n = jumlah record
= 10

$F_k + m = n + 1$ adalah rumus untuk mencari increament (angka pertambahan)
F_k = angka Fibonacci

$F_k + m = 10 + 1$
$= 11$
$m = 11 - F_k$

= 11 - 8 = 3

dimana 8 adalah bilangan Fib. terdekat dengan 11

Langkah-langkah :

Tentukan dulu F_{k-1}, F_{k-2} dan F_{k-3}

Bila F_k = 8 $\longrightarrow$ i = F_{k-1} = 5

p = F_{k-2} = 3

q = F_{k-3} = 2

Algoritme dalam Bahasa pemrograman C:

```
void fibonaci(void)
{  unsigned int fib[25];
   int I, p, q, m, j, t;
   int iterasi = 0;
   gettime(&waktu);
   printf("\n");
   cprintf("%d:%d:%d",waktu.ti_hour,waktu.ti_min,waktu.ti_sec);
   printf("\n");
   fib[0] = 0;
   fib[1] = 1;

   for( j = 2; j<15 ; j++)
       {   fib[j] = fib[j-1] + fib[j-2]; }
    printf("\n Fibonacci sequence : ");

    for( j = 0; j<15 ; j++)
        {   printf("%i ", fib[j]);}
    getch();

    j = 0;
    while ( fib[j] <= (n + 1))
          {  j= j+1; }
    m = n + 1 - fib[j-1];
    I = fib[j-2];
    p = fib[j-3];
    q = fib[j-4];
    printf("\n\n m= %i I= %i p= %i q=%i\n", m,I,p,q);
    getch();
```

```
 if ( nim K[I])
      { I = I + m;}
    while ( I != 0 )
       { iterasi = iterasi + 1;
       printf("\niterasi ke-%d I = %d K[%d] = %d",iterasi, I, I, K[I] );
       if ( nim < K[I] )
          { if (q != 0)
              { I = I-q;
                t = p;
                p = q;
                q = t-q;  }
            else
               I = 0; }
     else
         { if (nim < K[I])
              { if (p != 1)
                    { I = I+ q; p=p-q; q = q-p;}
                else
                    I = 1;}
           else {gettime(&waktu); printf("\n");
                  cprintf("%d:%d:%d",waktu.ti_hour,
                                        waktu.ti_min,waktu.ti_sec);
                  printf("\n");
                  return;}
               }
      }
}
```

jika X > K[i] maka i = i + m
ELSE
Lakukan WHILE - DO bila i < > 0 yang berisi
case - of : 1. X < K[i] : jika q = 0 maka i = 0
q <> 0 maka i = i - q;
t = p
p = q;
q = t-q

2. X = K[i] : RETURN

3. X > K[i] : jika p = 1 maka i = 0
p <> 1 ;
i = i + q;
p = p - q
q = q - p

contoh :

Cari record yang berharga 65 ⟶ X = 65
i = 5 ;
p = 3 ;
q = 2
m = 3

Cek: 65 > K[i] ⟶ ya, maka i = 5 + 3 = 8

masuk WHILE-DO :

case - 1 : Q < > 0 ⟶ i = i - q
= 8-2 =6
t = p = 3
p = q = 2
q = t - q
= 3-2 = 1

case-2 : 65 = K[6]

diperlukan 2 iterasi atau loop untuk mendapatkan 65.

4. Hash Table Search

Berbeda dengan search sebelumnya, pada *hash tables* search ini yang dicari dalam file bukan informasinya (tidak menggunakan identifier field) tapi mencari alamat atau lokasi dari identifier yang diberikan.

Misal akan dicari elemen yang berisi X, maka alamat X akan dihitung dulu menggunakan **FUNGSI HASH** (Hashing Function). Lokasi dari alamat tersebut terletak dalam **tabel HASH (HASH TABLE).** Hashing Function (fx), digunakan untuk melaksanakan transformasi pada suatu identifier X kedalam suatu alamat yang ada dalam *hash table*.

Contoh :

Hash table dengan 26 baris ('buckets') dan 2 kolom ('slot'). Identifier GA, D, G, L, A2, A1 dan E akan di 'hash' kedalam tabel.

Slot-1 slot-2

A1	A2
0	0
0	0
D	0
E	0
0	0
……	…….

f(A) = 1

f(D) = 4

f(E) = 5

f(G) = 7

dst.

karena hash table(1,1) sudah di Isi A1 maka A2 disimpan di (1,2)

Hash table bisa dibayangkan sebagai suatu matriks yang setiap barisnya mewakili identifier dari f(x). Diambil huruf pertama dari identifier X. Baris dibagi-bagi menjadi beberapa slot atau kolom dimana masing-masing kolom berisi 1 record.

Sort (Pengurutan)

Operasi pengurutan (sorting) adalah operasi yang sangat banyak dilakukan dalam "Data Processing".

Macam-macam Sort :

a. Selection Sort
b. Bubble Sort
c. Merge Sort
d. Quick Sort
e. Heap Sort

1. Selection Sort

Algoritme :

1. Ulangi langkah 2.sd.3 sebanyak n-1 kali
2. Cari indeks dari harga terkecil/terbesar
3. Tukarkan nilai indeks pertama dengan indeks terkecil/terbesar yang ada dalam elemen loop

UNSORTED PASS NUMBER(I) SORTED

j	k_j	1	2	3	4	5	6	7	8	9	
1	42	11	11	11	11	11	11	11	11	11	11
2	23	23	23	23	23	23	23	23	23	23	23
3	74	74	74	36	36	36	36	36	36	36	36
4	11	42	42	42	42	42	42	42	42	42	42
5	65	65	65	65	65	58	58	58	58	58	58
6	58	58	58	58	58	65	65	65	65	65	65
7	94	94	94	94	94	94	94	74	74	74	74
8	36	36	36	74	74	74	74	94	87	87	87
9	99	99	99	99	99	99	99	99	99	94	94
10	87	87	87	87	87	87	87	87	94	99	99

Algoritme Selection Sort dalam Bahasa pemrograman C:

```
void SELECTION_SORT(void)
{ PASS  = 1;
  do {  MIN_INDEX  = PASS;
        I := PASS;
          do { if (K[I] < K[MIN_INDEX])
                {
                    MIN_INDEX = I ;
                    I  =  I+ 1;
                 }
          while  ( I > N)

          if  ( MIN_INDEX != PASS)
             {
              X  =  K[PASS];
              K[PASS]  =  K[MIN_INDEX];
              K[MIN_INDEX]  =  X; }
              PASS  =  PASS  +  1;
            }
   while  (PASS  ==  N)
}
```

2. Bubble Sort

ALGORITME :

1. Repeat langkah 2 sampai dengan 4 sebanyak n-1 kali
2. Repeat langkah 3 untuk elemen yang belum sort
3. jika current elemen > next elemen, tukar
4. jika tidak ada pertukaran, return
   kalau tidak, kurangi ukuran dari elemen yang belum sort.

UNSORTED		PASS NUMBER(I)					SORTED
j	k_j	1	2	3	4	5	6
1	42	23	23	11	11	11	11
2	23	42	11	23	23	23	23
3	74	11	42	42	42	36	36
4	11	65	58	58	36	42	42
5	65	58	65	36	58	58	58
6	58	74	36	65	65	65	65
7	94	36	74	74	74	74	74
8	36	94	87	87	87	87	87
9	99	87	94	94	94	94	94
10	87	99	99	99	99	99	99

Algoritme dalam Bahasa pemrograman C:

```
void  BUBBLE_SORT(void) {
  LAST = N-1 ;
  PASS  =  1;

  do  {
        EXCHS  = 0;
        L  = 1;
         do  {
              if   (K[L]  > K[L+1])
                    {  X = K[L] ;
                       K[L] = K[L+1] ;
                       K[L+1]  =  X;
```

```
                    EXCHS  =  EXCHS + 1;
                }
                L  = L+1;   }
        while  (L > LAST)

        if  (EXCHS  !=  0)
          {
            LAST  =  LAST -1 ;
            PAST  =  PAST + 1;
          }
        else
            return(0):
while  (PASS  ==  N)
}
```

3. Merge Sort

Menggunakan konsep **"DIVIDE AND CONQUER"**

Algoritme :

1. gabungkan setiap kali 2 deretan angka dan lakukan sort terhadap gabungan angka tersebut.
2. Ulangi sampai diperoleh 2 deretan yang akan digabung menjadi 1 deretan angka yang sudah sort.

Procedure Merge sort dijalankan untuk data berikut ini :

42 23 74 11 65 58 94 36 99 87 divide

23 42 11 74 58 65 36 94 87 99 1. sort & Div

11 23 42 74 36 58 65 94 87 99 2 .sort & Div

11 23 36 42 58 65 74 94 87 99 3 .sort & Div

11 23 36 42 58 65 74 87 94 99 4 .sort

4. Quick Sort (Partition–Exchange Sort)

Sangat baik untuk **tabel yang sangat besar**.

Batas bawah dinyatakan dengan **LB** (Low Bond). **Batas atas** dinyatakan dengan **UB** (Upper Bound). Mempunyai algoritme sebagai berikut :

Algoritme dalam Bahasa pemrograman C:

```
void  QUICK_SORT(void)
{ Flag = " True";
  if  (LB < UB)
     { I  = LB;
       J  = UB + 1;
       KEY  = K[LB];
       while ( FLAG == "True")
             { I  = I + 1;
               while  ( K[I] < KEY)
                      { I  =  I + 1;
               }
               J  = J – 1;
               while  (K[J] > KEY)
                      { J  =  J  - 1;
                }
               if  (I<J) {
                     X =  K[I];
                     K[I]  =  K[J];
                     K[J]  :=  X;
               }
               else
                   FLAG  = FALSE;
        }
   X  =  K[J];
   K[LB]  =  K[J];
   K[J]    =  X;
   UB  = J-1;
    QUICK_SORT( );
    LB = J+1;
    QUICK_SORT( );
}
```

Misal diberikan data sebagai berikut :

	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	
(	42	23	74	11	65	58	94	36	99	87	)
(	11	23	36)	**42**	(65	58	94	74	99	87	)

11 (23 36) **42** (58) **65** (94 74 99 87)

11 **23** (36) **42** **58** **65** (94 74 99 87)

11 **23** **36** **42** **58** **65** (87 74) **94** (99)

11 **23** **36** **42** **58** **65** (74) **87** **94** **99**

hasilnya :

11 **23** **36** **42** **58** **65** **74** **87** **94** **99**

Algoritme didasarkan pada *Recursive Approach*

1. bagi tabel kedalam subtabel
2. gunakan QUICK_SORT dengan batas bawah untuk tabel kiri
3. gunakan QUICK_SORT batas atas untuk tabel kanan

5. Heap Sort

Langkah-langkah HEAP SORT :

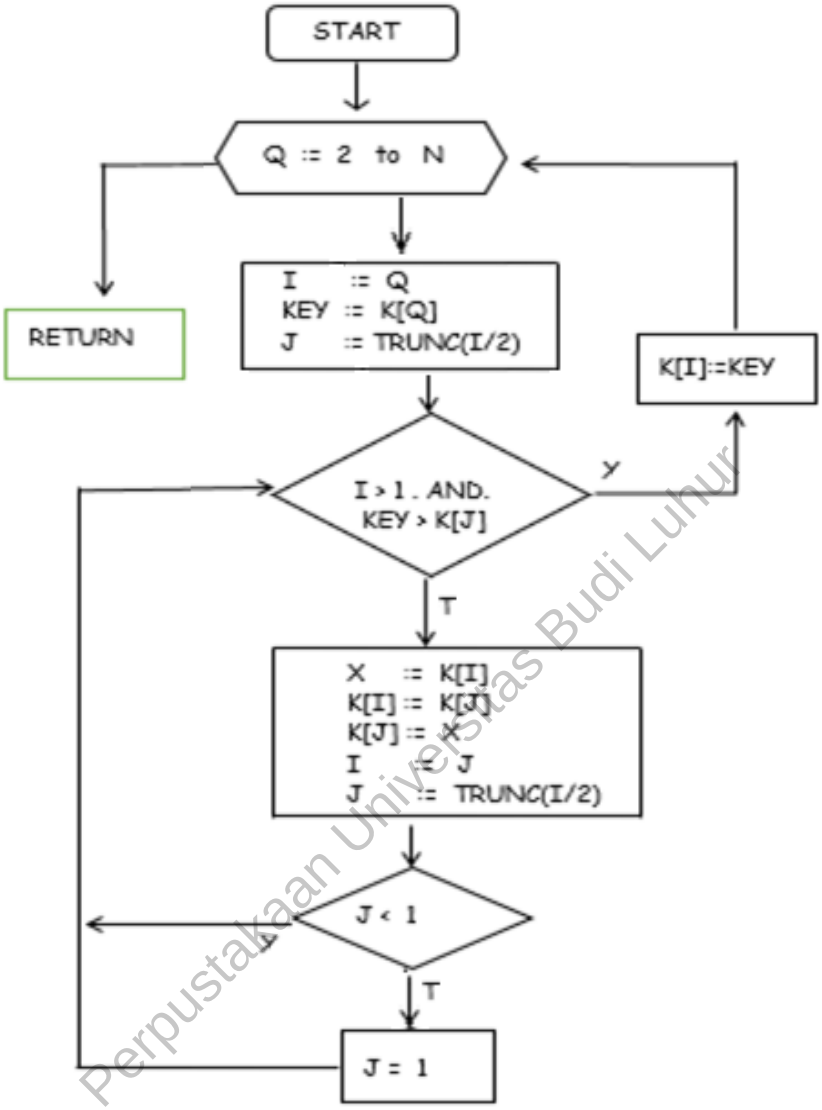
1. Create sebuah heap dengan data yang sudah ditentukan
2. Satu per satu pindahkan elemen yang paling besar dan bentuk kembali sebuah heap terhadap elemen-elemen sisanya

Pohon Heap adalah pohon yang punya sifat akar lebih besar dari anak kiri maupun anak kanan.

Misal input data sebagai berikut :

42 23 74 11 65 58 94 36 99 87

Pembentukan pohon heap, lihat langkah dalam flowchart dibawah ini



akan terbentuk urutan pohon heap sebagai berikut :

Q = 1

42

Q= 2

42

23

Q = 3

74

23 42

Q = 4

74

23 42

11 11

Q = 5

74

65 42

23

Q = 6

74

65 58

11 23 42

Q = 7

94

65 74

11 23 42 58 36

Q = 8

94

65 74

23 42 58

11

Q=9

99

94 74

65 23 42 58

11 36

Q = 10

99

94 74

65 87 42 58

11 36 23

Langkah-langkah pengurutan :
diambil nilai terbesar, dalam hal ini adalah nilai akar

Q = 10 : K[10] = 99

94

87 74

65 23 42 58

11 36

Q=9 : K[9] = 94

87

65 74

36 23 42 58

11

Q=8: K[8] = 87

74

65 58

36 23 42 11

Q=7: K[7] = 74

65

36 58

11 23 42

Q = 6: K[6] = 65

58

35 42

11 23

Q=5: K[5] = 42

42

35 23

11

Q=4: K[4] = 42

36

11 23

Q=3: K[3] = 23

23

11

Q=2: K[2] = 23

11

Sehingga diperoleh urutan dalam file - K sebagai berikut:

11 23 36 42 58 65 74 87 94 99

Algoritme HEAP_SORT(K, N) :

1. Bentuk pohon heap dengan procedure yang ada
2. Laksanakan sort
   ulangi sampai dengan langkah 10; Q = N, N-1, ... , 2
   Tukarkan nilai record ; K[I] ←——→ K[Q]
3. Berikan harga awal
   I = 1
   KEY = K[I]
   J = 2
4. Dapatkan index dari harga terbesar anak dari record baru
   if ((J+1) < Q)
   { if (K[J+1] > K[J])
   {J = J+1; } }
5. Bangun kembali heap baru
6. Ulangi sampai dengan langkah 10 apabila
   (J < (Q-1)) && (K[J] > KEY)
   Tukarkan harga record : K[I] ←——→ K[J]
7. Dapatkan anak kiri berikutnya
   I = J;
   J = J + 1;
8. Dapatkan index dari anak dengan harga terbesar berikutnya
   if ((J+1) < Q }
   { if (K[J+1] > K[J])
   { J = J+1;}
   else {
   if (J > N)
   { J = N; } }
9. Copykan record kedalam tempatnya yang cocok
   K[I] = KEY
10. return

Program Sort dalam bahasa C:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#define   n   10

void selection_sort(void);
void bubble_sort(void);
void quick_sort();
void gettime(struct time *pwaktu);
static  int K[n+1] = { 0, 42, 23, 74, 11, 65, 58, 94, 36, 99, 87};
int  nim, p;
int  ulang = 0;

main() {
   struct time waktu;
   int  pilih, k;
   int  lb=1, ub=10;
   ClrBarloop:
   {
   clrscr();
   printf("\n");
   gotoxy(33,2); puts("Coba  Sort");
   gotoxy(18,3); puts("lakukan exit dulu untuk pilihan yang lain");
   gotoxy(28,6); puts(" 1.  Selection  Sort");
   gotoxy(28,8); puts(" 2.  Bubble  Sort");
   gotoxy(28,10); puts(" 3.  Quick  Sort");
   gotoxy(28,12); puts(" 4.  exit");
   gotoxy(33,15); printf("Pilih : ");
   gotoxy(41,15); scanf("%x",  &pilih);

   switch(pilih)   {
   case  1:  printf("\n");  gettime(&waktu);
             cprintf("%2d:%2d:%2d", waktu.ti_hour, waktu.ti_min,waktu.ti_sec);
              printf("\n");
              selection_sort();
              printf("\n\n");
```

```
            gettime(&waktu);
            cprintf("%2d:%2d:%2d", waktu.ti_hour, waktu.ti_min,waktu.ti_sec);
            getch();
            break;

   case  2: printf("\n");     gettime(&waktu);
            cprintf("%2d:%2d:%2d", waktu.ti_hour, waktu.ti_min,waktu.ti_sec);
            printf("\n");
            bubble_sort();
            printf("\n\n");
            gettime(&waktu);
            cprintf("%2d:%2d:%2d", waktu.ti_hour, waktu.ti_min,waktu.ti_sec);
            getch();
            break;

   case  3:  printf("\n");  gettime(&waktu);
            cprintf("%2d:%2d:%2d", waktu.ti_hour, waktu.ti_min,waktu.ti_sec);
            printf("\n");
            quick_sort (K, lb,  ub);
            printf("\n\n");
            for( k = 1; k <= n; k++)
                  {   printf(" %i  ", K[k]);  }
            printf("\n\n");
            getch();
            gettime(&waktu);
            cprintf("%2d:%2d:%2d", waktu.ti_hour, waktu.ti_min,waktu.ti_sec);
            getch();
            break;
   case  4:  exit(0);
            break;
   }
  goto ClrBarloop;
  }
}

void selection_sort(void)
 { int  pass, min_index, I , x;
   int  iterasi=0;
   pass = 1;
   do
     { min_index = pass;
```

```
      I = pass;
      do
          { if (K[I] < K[min_index])  {
           min_index = I;}
           I = I + 1;
     }
    while (I <=  n);
    if (min_index  !=  pass)
         {  x = K[pass];
           K[pass] = K[min_index];
           K[min_index]= x;  }
     pass = pass + 1; iterasi = iterasi + 1;
     }

  while (pass < n );
   printf("\n data  sort  :\n");
   for( pass = 1; pass <= n; pass ++)
          {printf(" %i  ", K[pass]);}
   printf("\n\n iterasi = %d ", iterasi);
 }

void bubble_sort(void)
 { int  last,  pass, exchs, L;
   int  x, iterasi,j;
   last  =  n;
   pass  = 1;
   iterasi = 0;

   do
   { exchs = 0;
     iterasi = iterasi + 1;
     L  = 1;
     do
     {  if  (K[L]  >  K[L+1])
             {  x = K[L];
                K[L] = K[L+1];
                K[L+1] = x;
                exchs = exchs + 1;      }
          L = L+1;
     }
    while  (L < last);
```

```
        printf("\niterasi ke - %i   :  ", iterasi);
       for( j = 1; j < n; j++)
            {  printf(" %i  ", K[j]);  }
             getch();
    if (exchs ==  0)
         {break;}
    else
        { last = last -1;
          pass = pass + 1; }
   }
   while (pass < n);
   printf("\n data  sort  :\n");
   for( pass = 1; pass < n; pass ++)
         {printf(" %i  ", K[pass]);}
 }

void quick_sort(int K[], int lb, int ub)
  { char flag = '1';
    int  bawah = lb, atas = ub;
    int  I, J, Key,  X;
    int  j;
    if (bawah <  atas)
         { I = bawah;
           J= atas + 1;
           Key = K[bawah];
          ulang = ulang + 1;

          while (flag == '1')
          {  I = I+1;
             while (K[I] < Key)
                    { I =  I + 1;}
             J = J-1;
            while  (K[J] > Key)
                     {J = J - 1;}
            if (I < J)
              { X = K[I];
                K[I] = K[J];
                K[J] = X;}
            else
              { flag = '0';}
          }
```

```
    X = K[J];
    K[J]= K[bawah];
    K[bawah] = X;

    printf("\niterasi ke - %i :   partisi = K[%i]= %i ", ulang,J,K[J]);
    printf("\n                  bawah   :");

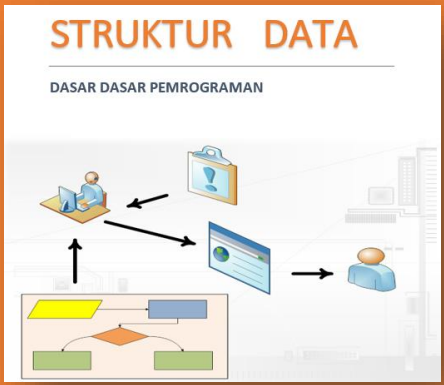
    for( j = bawah; j <= (J-1); j++)
      {printf(" %i  ", K[j]);}
    getch();   printf("\n                 atas    :");

    for( j = (J+1); j <= atas; j++)
      {printf(" %i  ", K[j]);}
    getch();

    quick_sort(K,bawah,J-1);
    quick_sort(K,J+1,atas);
    }
}
```

Daftar Pustaka

1. Horowithz, Ellis and Sartaj Sahni, "*Fundamental of Computer Algorithms*", Computer Science Press, 1988
2. Robert Sedgewick, "*Algorithms in C++*", Addison-Wesley Publishing Company, 1992
3. Robert L. Kruse, Bruce P.Leung, Clouis L. Tondo, "*Data Structures and Program Design in C*", Prentice Hall 1997
4. Mark Allen Weiss, "*Data Structure & Algorithm Analysis in C++*" , The Benjamin/Cummings Publishing Company, Inc., 1994
5. William Ford and William Topp, "*Data Dtructures with C++*" , Prentice Hall, 1997



Buku ini cocok digunakan untuk mahasiswa, pelajar atau pemula yang bermaksud untuk mempelajari dasar-dasar struktur data dan implementasinya. Jenis jenis tipe data serta operasinya, array, linked list, stack, queue pengurutan (sort) dan pencarian (search). Contoh implementasi menggunakan bahasa pemrograman Bahasa C.

Dasar-dasar dari algoritme untuk materi teknologi komputer, seperti artificial intelligence, Expert system, Data minning, algoritme dan pemrograman.

Sebagai manusia yang tidak luput dari kekurangan dan kesalahan, kami sadar bahwa buku ini masih jauh dari sempurna, oleh karena itu masukan-masukan diharapkan agar dapat melengkapi buku ini

PENERBIT
UNIVERSITAS BUDI LUHUR

ISBN 978-623-92135-3-4
9 786239 213534