Diktat

STRUKTUR DATA

Oleh :

R. Joko Musridho, S.T., M.Phil.

NIDN : 1021109102

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS PAHLAWAN TUANKU TAMBUSAI

2024

KATA PENGANTAR

Bismillahirrahmanirrahim...

Alhamdulillah wa syukurillah penulis panjatkan puji syukur kepada Allah SWT berkat rahmat, pengetahuan dan karuniaNya penulis dapat menuliskan dan menghasilkan Diktat Edisi Revisi untuk matakuliah Struktur Data.

Matakuliah ini adalah salah satu matakuliah yang diajarkan pada kurikulum Prodi Teknik Informatika di Fakultas Teknik Universitas Pahlawan Tuanku Tambusai dan diajarkan pada kelas semester 2.

Penulis mengucapkan banyak terimakasih atas dukungan dan bantuan para pimpinan, rekan-rekan dosen, teman sejawat di lingkungan Fakultas Teknik serta Universitas Pahlawan Tuanku Tambusai atas terselesaikannya Diktat ini. Semoga Diktat ini dapat membantu dan mendukung PBM di Prodi Teknik Informatika.

Penulis juga menyadari masih adanya kekurangan dan keterbatasan pada Diktat ini, maka penulis tetap mengharapkan kritik dan saran dari berbagai pihak agar Diktat ini bisa dikembangkan dikemudian hari. Akhir kata semoga segala upaya yang penulis lakukan ini bermanfaat bagi kita semua dan Semoga Allah SWT berkenan memberikan berkahnya sehingga semua harapan dan cita-cita penulis dapat terkabulkan. Amin

Bangkinang, Januari 2024

R. Joko Musridho, S.T., M.Phil.

DAFTAR ISI

Halaman

BAB I
KONSEP ABSTRAC TYPE DATA dan STRUC

A. Pengertian Struktura Data

Struktur data adalah cara penyimpanan, pengorganisasian, dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien.[1] Sedangkan data adalah representasi dari fakta dunia nyata. Fakta atau keterangan tentang kenyataan yang disimpan, direkam atau direpresentasikan dalam bentuk tulisan, suara, gambar, sinyal atau simbol.

Secara garis besar type data dapat dikategorikan menjadi :

1. Type data sederhana
    a. Type data sederhana tunggal, misalnya
        Integer, real, boolean dan karakter
    b. Type data sederhana majemuk, misalnya
        String
2. Struktur Data, meliputi
    a. Struktur data sederhana, misalnya array dan record
    b. Struktur data majemuk, yang terdiri dari
        Linier : Stack, Queue, serta List dan Multilist
        Non Linier : Pohon Biner dan Graph

Pemakaian struktur data yang tepat di dalam proses pemrograman akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana.

Struktur data yang ″standar″ yang biasanya digunakan dibidang informatika adalah :

- List linier (Linked List) dan variasinya Multilist
- Stack (Tumpukan)
- Queue (Antrian)
- Tree (Pohon)
- Graph (Graf)

[1] Moh, Sjukani,2012,Struktur Data (Algoritma dan Struktur Data dengan C,C++),Jakarta:Mitra Wacana Media

B. Tipe Data Abstrak

Tipe data abstrak (ADT) dapat didefinisikan sebagai *model matematika dari objek data yang menyempurnakan tipe data dengan cara mengaitkannya dengan fungsi-fungsi yang beroprasi pada data yang bersangkutan*.[2] Merupakan hal yang sangat penting untuk mengenali bahwa operasi-operasi yang akan dimanipulasi data pada objek yang bersangkutan termuat dalam spesifikasi ADT. Sebagai contoh, ADT HIMPUNAN didefinisikan sebagai koleksi data yang diakses oleh operasi-operasi himpunan seperti penggabungan (UNION), irisan (INTERSECTION), dan selisih antar-himpunan (SET DIFFERENCE).

Implementasi dari ADT harus menyediakan cara tertentu untuk merepresentasikan unsur tipe data (seperti matrix) dan cara untuk mengimplementasikan operasi - operasi matrix. Secara tipikal, kita akan mendeskripsikan operasi-operasi pada ADT dengan **algoritma** (logika berfikir) tertentu. Algoritma ini biasanya berupa urutan instruksi yang menspesifikasi secara tepat bagaimana operasi-operasi akan dilakukan/dieksekusi oleh komputer.[3]

ADT adalah tipe data tertentu yang didefinisikan oleh pemrogram untuk kemudahan pemrograman serta untuk mengakomodasi tipe-tipe data yang tidak secara spesifik diakomodasi oleh bahasa pemrograman yang digunakan. Maka, secara informal dapat dinyatakan bahwa ADT adalah :

1. Tipe data abstrak ADT pertama kali ditemukan oleh para ilmuan komputer utuk memisahkan struktur penyimpanan dari perilaku tipe data yang abstrak seperti misalnya, Tumpukan(Stack) serta antrian(Queue). Seperti kita duga, pemrogram tidak perlu tahu bagaimana Tumpukan(Stack) perubahan inplementasi ADT tidak mengubah program yang menggunakannya secara keseluruhan, dengan catatan bahwa interface ADT tersebut dengan ‘dunia luar’ tetap dipertahankan.

[2] Wirth,1986, N.Algorithms & Data Stuctures, Prentice Hall
[3] Aho, Hopcroft, Ullman,1987,Data Structures and Algorithms,Prentice Hall

2. Pemakaian dan pembuatan ADT dapat dilakukan secara terpisah. yang perlu dibicarakan antara pembuat dan pengguna ADT adalah *interface* ADT yang bersangkutan.
3. ADT merupakan sarana pengembangan sistem yang bersifat *modular*, memungkinkan suatu sistem dikembangkan oleh beberapa orang anggota tim kerja dimana masing-masing anggota tim bisa melakukan bagiannya sendiri-sendiri dengan tetap mempertahankan keterpaduannya dengan anggota tim yang lain.[4]

Dalam hal ini perlu dibedakan antara pengertian **struktur data** dan ADT. Struktur data hanya memperlihatkan bagaimana data-data di organisir, sedangkan ADT bercakupan lebih luas, yaitu memuat/mengemas struktur data tertentu sekaligus dengan operasi-operasi yang dapat dilakukan pada struktur data tersebut. Dengan demikian, definisi umum tentang ADT di atas dapat diperluas sebagai berikut :

Implementasi ADT={Struktur Data(Operasi-operasi yang Dapat Dilakukan Terhadap Struktur Data)}

Bahasa pemrograman bisa memiliki tipe data:

- **Built-in** : sudah tersedia oleh bahasa pemrograman tersebut
  - Tidak berorientasi pada persoalan yang dihadapi.
- **UDT** : *User Defined Type*, dibuat oleh pemrogram.
  - Mendekati penyelesaian persoalan yang dihadapi.
  - Contoh: *record* pada Pascal, *struct* pada C, *class* pada Java.
- **ADT** : *Abstract Data Type*,
  - memperluas konsep UDT dengan menambahkan pengkapsulan atau enkapsulasi, berisi sifat-sifat dan operasi-operasi yang bisa dilakukan terhadap kelas tersebut.

[4] Aho, Hopcroft, Ullman,1987,Data Structures and Algorithms,Prentice Hall

Bahasa C++ memiliki tipe data numerik dan karakter (seperti *int*, *float*, *char* dan lain-lain). Disamping itu juga memiliki tipe data *enumerasi* dan *structure*. Bagaimana jika kita ingin membuat tipe data baru?

- Untuk pembuatan tipe data baru digunakan keyword **typedef**

↗ **Bentuk umum:**

- typedef <tipe_data_lama> <nama_tipe_data_baru>

Program :

```
#include <iostream>
using namespace std;
typedef int angka;
typedef float pecahan;
typedef char huruf;
int main()
{
	huruf nama[10];
	angka umur;
	huruf h;
	pecahan pecah;
	cout<<"Masukkan Nama Anda = ";
	cin>>nama;
	cout<<"Masukkan Umur Anda = ";
	cin>>umur;
	cout<<"Masukkan sembarang huruf = ";
	cin>>h;
	cout<<"Masukkan sembarang bilangan pecahan = ";		cin>>pecah;
	cout<<"Nama Anda = "<<nama<<endl;
	cout<<"Umur Anda = "<<umur<<endl;
	cout<<"Huruf Sembarang = "<<h<<endl;
	cout<<"Pecahan Sembarang = "<<pecah<<endl;
	return 0;
}
```

C. STRUCT

Struct adalah tipe data bentukan yang berisi kumpulan variabel-variabel yang bernaung dalam satu nama yang sama dan memiliki kaitan satu sama lain.[5] Berbeda dengan *array* hanya berupa kumpulan variabel yang bertipe data sama, *struct* bisa memiliki variabel-variabel yang bertipe data sama atau berbeda, bahkan bisa menyimpan variabel yang bertipe data *array* atau *struct* itu sendiri. Variabel-variabel yang menjadi anggota *struct* disebut dengan *elemen struct*.

1. Bentuk umum :

```
struct <nama_struct> {
        tipe_data <nama_var>;
        tipe_data <nama_var>;
        ....
} [6]
```

2. Cara mendeklarasikan Struct

```
struct Mahasiswa {
char NIM[8];
char nama[50];
float ipk;
};
```

Keterangan deklarasi struct

- Untuk menggunakan *struct* Mahasiswa dengan membuat variabel mhs dan mhs2

  Mahasiswa mhs, mhs2;
- Untuk menggunakan struct Mahasiswa dengan membuat variabel *array* m;

  Mahasiswa m[100];

3. Cara Menggunakan Struct

Penggunaan/pemakaian tipe data *struct* dilakukan dengan membuat suatu variabel yang bertipe data *struct* tersebut. Pengaksesan elemen *struct* dilakukan secara individual dengan menyebutkan nama variabel *struct* diikuti dengan

[5] Hariyanto, Bambang,2000,Struktur Data,Bandung

[6] Dwi, Sanjaya,,2005,Asyiknya Belajar Struktur Data di Planet C++,Jakarta:Elex Media Komputindo,Jakarta

operator titik (.). Misalnya dengan *struct* mahasiswa seperti contoh di atas, kita akan akses elemen-elemennya seperti contoh berikut:

Program :

```
#include <iostream>
#include <stdio.h>
using namespace std;
struct Mahasiswa
{
   char NIM[9];
   char nama[30];
   float ipk; };
int main()
{
   Mahasiswa mhs;
   system("cls");
   printf("NIM = ");scanf("%s", &mhs.NIM);
   printf("Nama = ");scanf("%s", &mhs.nama);
   printf("IPK = ");scanf("%f", &mhs.ipk);
   printf("Data Anda : \n");
   printf("NIM : %s\n", mhs.NIM);
   printf("Nama : %s\n", mhs.nama);
   printf("IPK : %f\n", mhs.ipk);
   return 0;
}
```

Latihan

Seorang pegawai perusahaan mempunyai gaji pokok Rp. 3.500.000/bulan, Tunjangan Istri Rp. 300.000, Tunjangan Anak Rp. 250.000. Setiap bulan wajib membayar iuran koperasi sebesar Rp. 200.000. Setiap pegawai dikenakan pajak penghasilan sebesar 5% dan BPJS 1% dari gaji bersih pegawai.

Buatlah program menggunakan *struct* dari kasus tersebut, Carilah total gaji setiap bulan!

BAB II
ARRAY

Array merupakan bagian dasar pembentukan suatu struktur data yang lebih kompleks.[7] Hampir setiap jenis struktur data kompleks dapat disajikan secara logik oleh array.

- **Array** : Suatu himpunan hingga elemen yang terurut dan homogen, atau dapat didefinisikan juga sebagai pemesanan alokasi memory sementara pada komputer.
- **Terurut** : elemen tersebut dapat diidentifikasi sebagai element pertama, kedua, dan seterusnya sampai elemen ke-n.
- **Homogen** : setiap elemen data dari sebuah array tertentu haruslah mempunyai tipe data yang sama.

Karakteristik Array :

1. Mempunyai batasan dari pemesanan alokasi memory (bersifat statis)
2. Mempunyai type data sama (bersifat Homogen)
3. Dapat diakses secara acak.
4. Berurutan (terstruktur).

Array mempunyai dimensi :

1. Array Dimensi Satu (Vektor)
2. Array Dimensi Banyak.
    - Dimensi Dua (Matriks/Tabel)
    - Dimensi Tiga (Kubik).

A. ARRAY DIMENSI SATU

Merupakan bentuk yang sangat sederhana dari array. Setiap elemen array mempunyai subskrip/indeks. Fungsi indeks/subskrip ini antara lain :

1. Menyatakan posisi elemen pada array tsb.
2. Membedakan dengan elemen lain.

Penggambaran secara fisik Array A(1:N) :

[7] Horowitz, E. & Sahni, S,1984, Fundamentals of Data Structures in Pascal,Pitman Publishing Limited.

A(1)	A(2)	A(3)	A(4)	…	A(N)

Ket : A : nama array

1,2,3,4,…,N : indeks / subskrip

- Secara umum Array Dimensi Satu A dengan tipe T dan subskrip bergerak dari L sampai U ditulis : **A(L:U) = (A(I)); I =L , L+1, L+2, …, U**

  Keterangan : L : batas bawah indeks / lower bound

  U : batas atas indeks / upper bound

  A : nama Array
- Banyaknya elemen array disebut **Rentang atau Range A(L:U) = U – L + 1**
- Range khusus untuk array Dimensi Satu yang mempunyai batas bawah indeks L=1 dan batas atas U=N, maka **Range A adalah A(1:N) =(N – 1) + 1 = N**

Contoh :

Data hasil pencatatan suhu suatu ruangan setiap satu jam, selama periode 24 jam ditulis dalam bentuk Array Dimensi Satu menjadi

Misal :

nama arraynya Suhu, berarti elemennya dapat kita tulis sebagai Suhu(I), dengan batas bawah 1 dan batas atas 24.

Suhu(I):menyatakan suhu pada jam ke-I dan 1<=I<=24

Range Suhu(1:24)=(24-1)+1=24

```
 1  #include <iostream>
 2  #include <stdio.h>
 3
 4  using namespace std;
 5
 6  int main()
 7  {
 8       int a[8];
 9       int i;
10
11       for (i=0;i<8;i++){
12            printf("%x \n", &a[i]);
13       }
14
15  }
```

B. ARRAY DIMENSI DUA

Array Dimensi Dua perlu dua subskrip/indeks , Indeks pertama untuk menyatakan posisi baris, Indeks kedua untuk menyatakan posisi kolom.[8]

- Secara umum Array Dimensi Dua B dengan elemen-elemen bertype data T dinyatakan sbb : **B(L1:U1,L2:U2)={B(I,J)}**

  L1<=1<=U1, L2<=1<=U2, dan setiap elemen B(I,J) bertype data T

  Keterangan : B = nama array

  L1 = batas bawah indeks baris

  L2 = batas bawah indeks kolom

  U1 = batas atas indeks baris

  U2 = batas atas indeks kolom

- Jumlah elemen baris dari array B adalah **(U2 - L2 + 1)**
- Jumlah elemen kolom dari array B adalah **(U1 – L1 + 1)**
- Jumlah total elemen array adalah **(U2 – L2 + 1) * (U1 – L1 + 1)**
- Suatu array B yang terdiri atas M elemen dimana elemennya berupa array dengan N elemen, maka dapat digambarkan sbb:

	1	2	3	……	J	…..	N
1							
2							
3							
…							
I					B(I,J)		
…							
M							

- Array diatas dituliskan :

  B(1:M, 1:N) = B(I,J) ;

  Untuk I = 1,2,3,…….,M dan

  J = 1,2,3,……., N

- Jumlah elemen array B : M * N

[8] Horowitz, E. & Sahni, S,1984, Fundamentals of Data Structures in Pascal, Pitman Publishing Limited

- Array B berukuran / berorder : M * N

```
1   #include <iostream>
2   #include <stdio.h>
3
4   using namespace std;
5
6   int main()
7   {
8       int a[3][5];
9
10      for (int i=0;i<3;i++){
11          for (int j=0;j<5;j++){
12              printf("%x ", &a[i][j]);
13          }
14          printf("\n");
15      }
16
17      return 0;
18  }
```

C. ARRAY DIMENSI TIGA

Banyaknya indeks yang diperlukan array dimensi tiga adalah 3. Pada umumnya, suatu array berdimensi N memerlukan N indeks untuk setiap elemennya.

➢ **Secara acak array berdimesi N ditulis sbb:**

A(L1:U1, L2:U2, …….., LN:UN) = (A(I1,I2,……,IN)) dengan L_K <= I_K<=U_K , k = 1,2,3,…, N[9]

Contoh :

Penyajian data mengenai jumlah mahsiswa Sistem Informasi Universitas Islam Negeri Medan berdasarkan tingkat, untuk kelas pagi dan malam dan jenis kelamin.

Jawab :

MHS = nama array

I = 1,2,3,4,5 (tingkat 1/5)

[9] Horowitz, E. & Sahni, S,1984, Fundamentals of Data Structures in Pascal, Pitman Publishing Limited

J = 1,2 (1 = pagi; 2 = malam)

K = 1,2 (1 = pria; 2= wanita)

MHS (1:5, 1:2,1:2)

Jadi :

- MHS(3,2,2)

jumlah mahasiswa tingkat 3 MI kelas malam untuk jenis kelamin wanita

- Cross Section MHS (1,*,2)

jumlah mahasiswa tingkat 1 MI kelas pagi atau malam dan berjenis kelamin wanita.

D. ARRAY MULTI DIMENSI

Memori komputer linier, maka untuk memetakan array dimensi banyak ke storage harus dilinierkan.[10]

Alternatif untuk pelinieran tersebut adalah :

- **Row Major**

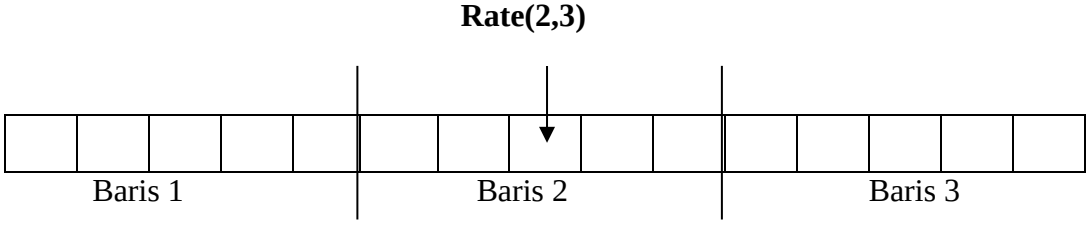
Biasanya digunakan COBOL, PASCAL

Menyimpan pertama kali baris pertama dari array, kemudian baris kedua, ketiga dst

Array Rate (1:3 , 1:5)

	1	2	3	4	5
1					
2			Rate(2,3)		
3					

Menjadi



[10] Horowitz, E. & Sahni, S,1984, Fundamentals of Data Structures in Pascal, Pitman Publishing Limited

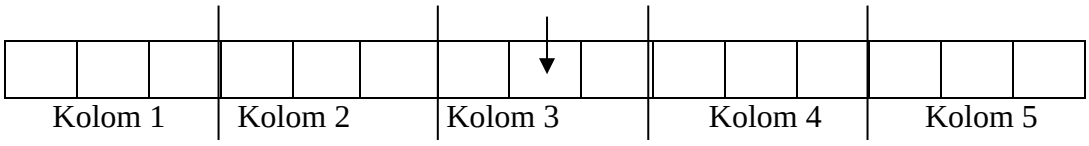
- Array A(I,J) dari array yang didefinisikan sebagai array
  A(L1:U1 , L2:U2), mempunyai
  Address awal : B + (I – L1) * (U2 – L2 +1) * S + (J – L2) * S

Contoh :

- Array A(1:3, 1:5) dan elemen A(2,3) mempunyai address awal :
  B + (2-1) * (5-1+1) * S + (3-1) * S
  B + 5 * S + 2 * S
  B + 7 * S
- Array A(2:4, 3:5) dan elemen A(3,4) mempunyai address awal :
  B + (3-2) * (5-3+1) * S + (4-3) * S
  B + 1 * 3 * S + 1 * S
  B + 4 * S

- **Column Major**

  Biasanya digunakan FORTRAN
  Menyimpan Kolom pertama dari array kemudian kolom kedua, ketiga, dst
  Menjadi



- Array A(I,J) dari array yang didefinisikan sebagai array
  A(L1:U1 , L2:U2) mempunyai
  Address awal : B + (J – L2) * (U1 – L1 +1) * S + (I – L1) * S

Contoh :

- Array A(1:3, 1:5) dan elemen A(2,3) mempunyai address awal :

  B + (3-1) * (3-1+1) * S + (2-1) * S

  B + 6 * S + 1 * S

  B + 7 * S

- Array A(2:4, 3:5) dan elemen A(3,4) mempunyai address awal :

  B + (4-3) * (4-2+1) * S + (3-2) * S

  B + 1 * 3 * S + 1 * S

  B + 4 * S

E. ARRAY SEGITIGA (TRINGULAR ARRAY)

Ada 2 macam

1. Upper Tringular

   Elemen dibawah diagonal utama adalah 0

2. Lower Tringular

   Elemen diatas diagonal utama adalah 0

- Suatu array dimana elemen diagonalnya juga nol disebut **Strictly (upper/lower) Tringular.**
- Pada array Lower Tringular dengan N baris, jumlah maksimum elemen <> 0 pada baris ke-I adalah I [11]
- **Total elemen <> 0 adalah** $\sum_{K=1}^{N} I = (N * (N+1)) /2$
- Untuk N kecil : tidak ada masalah
- Untuk N besar :
  1. Elemen yang sama dengan nol tidak usah disimpan dalam memori
  2. Pendekatan terhadap masalah ini adalah dengan pelinieran array dan hanya menyimpan array yang tidak nol.

[11] Wirth, N,1986,Algorithms & Data Stuctures, Prentice Hall

BAB III
SORTING ARRAY

Sorting merupakan suatu proses untuk menyusun kembali himpunan obyek menggunakan aturan tertentu. Sorting disebut juga sebagai suatu algoritma untuk meletakkan kumpulan elemen data kedalam urutan tertentu berdasarkan satu atau beberapa kunci dalam tiap-tiap elemen. [12]

Pada dasarnya ada dua macam urutan yang biasa digunakan dalam suatu proses sorting:

1. Urut naik (*ascending*) Mengurutkan dari data yang mempunyai nilai paling kecil sampai paling besar.
2. Urut turun (*descending*) Mengurutkan dari data yang mempunyai nilai paling besar sampai paling kecil.

A. Sorting Data

Ada banyak alasan dan keuntungan dengan mengurutkan data. Data yang terurut mudah untuk dicari, mudah untuk diperiksa, dan mudah untuk dibetulkan jika terdapat kesalahan. Data yang terurut dengan baik juga mudah untuk dihapus jika sewaktu-waktu data tersebut tidak diperlukan lagi. Selain itu, dengan mengurutkan data maka kita semakin mudah untuk menyisipkan data atapun melakukan penggabungan data

Metode pada sorting array :

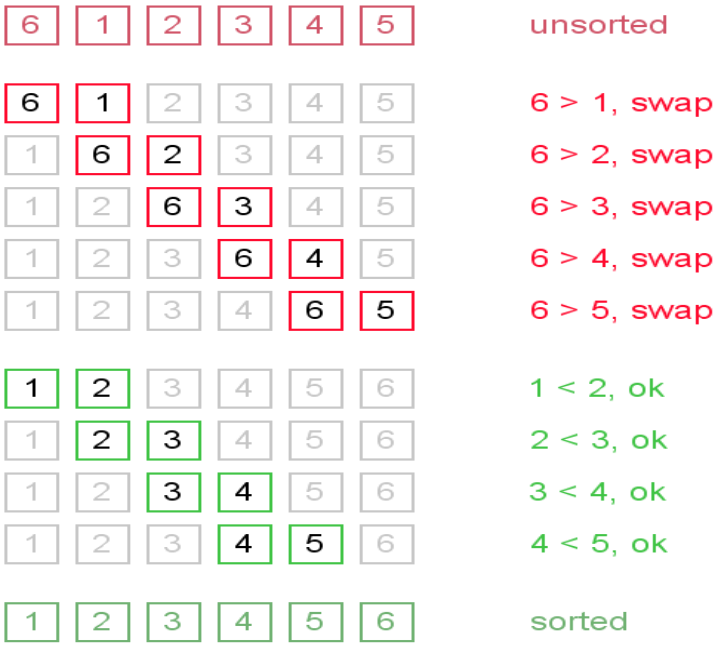
1. Pengurutan berdasarkan perbandingan (*comparison-based sorting)*
    - Bubble sort, Exchange sort
2. Pengurutan berdasarkan prioritas (*priority queue sorting method)*
    - Selection sort, Heap sort (menggunakan *tree*)
3. Pengurutan berdasarkan penyisipan dan penjagaan terurut (*insert and keep sorted method*)
    - Insertion sort, Tree sort
4. Pengurutan berdasarkan pembagian dan penguasaan (*devide and conquer method*)
    - Quick sort, Merge sort

[12] Hariyanto, Bambang,2000,Struktur Data,Bandung

5. Pengurutan berkurang menurun (*diminishing increment sort method*)
    - Shell sort (pengembangan *insertion*)
6. Pengurutan berdasarkan posisi dalam angka
    - Radix Sort[13]

B. Bubble Sort

Adalah metode sorting termudah, diberi nama *"Bubble"* karena proses pengurutan secara berangsur-angsur bergerak/berpindah ke posisinya yang tepat, seperti gelembung yang keluar dari sebuah gelas bersoda. *Bubble Sort* mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya.



Pengurutan *Ascending*:Jika elemen sekarang **lebih besar** dari elemen berikutnya maka kedua elemen tersebut **ditukar**. Pengurutan *Descending*: Jika elemen sekarang **lebih kecil** dari elemen berikutnya, maka kedua elemen tersebut **ditukar.** Algoritma ini seolah-olah menggeser satu per satu elemen dari kanan ke kiri atau kiri ke kanan, tergantung jenis pengurutannya, asc atau desc. Ketika satu proses telah selesai, maka *bubble sort* akan mengulangi proses, demikian seterusnya sampai dengan iterasi sebanyak n-1. Kapan berhentinya? *Bubble sort*

[13] Sjukani, Moh,Struktur Data (Algoritma dan Struktur Data dengan C,C++),Jakarta:Mitra Wacana Media,2012

berhenti jika seluruh array telah diperiksa dan tidak ada pertukaran lagi yang bisa dilakukan, serta tercapai perurutan yang telah diinginkan.

Cara kerjanya adalah dengan berulang-ulang melakukan traversal (*proses looping*) terhadap elemen-elemen struktur data yang belum diurutkan. Di dalam traversal tersebut,nilai dari dua elemen struktur data dibandingkan. Jika ternyata urutannya tidak sesuai dengan "pesanan",maka dilakukan pertukaran (swap). Algoritma sorting ini disebut juga dengan comparison sort dikarenakan hanya mengandalkan perbandingan nilai elemen untuk mengoperasikan elemennya. Algoritma Bubble Sort Algoritma bubble sort dapat diringkas sebagai berikut, jika N adalah panjang elemen struktur data, dengan elemen-elemennya adalah T1, T2, T3, …, TN-1,TN.

maka:

1. Lakukan traversal untuk membandingkan dua elemen berdekatan. Traversal ini dilakukan dari belakang.
2. Jika elemen pada TN-1 > TN , maka lakukan pertukaran (swap). Jika tidak, lanjutkan ke proses traversal berikutnya sampai bertemu dengan bagian struktur data yang telah diurutkan.
3. Ulangi langkah di atas untuk struktur data yang tersisa

Program :

IMPLEMENTASI 1 ASCENDING

--

```
void bubble_sort1(){
        for (int i=1; i<n; i++){
                for(int j=n-1; j>i; j--){
                        if(data[j] < data[j-1]) tukar(j, j-1);
                }
        }
}
```

IMPLEMENTASI 2 DESCENDING

--

```
void bubble_sort2(){
        for (int i=1; i<n; i++){
```

```
                for(int j=0; j<n-i; j++){
                        if(data[j] < data[j+1]) tukar(j, j+1);
                }
        }
}
```

Program Bubble Sort

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>

void bubble_sort (int array[], int size)
{int temp,i, j;

for (i=0; i<size-1; i++)
   for (j=0; j<size-1-i; j++)
      if (array[j] > array[j+1])
      {
      temp= array [j];
      array[j] = array[j+1];
      array [j+1]=temp;
      }
}
main()
{
int values[30],i;

clrscr();
//apa aja blh
cout << “data yang belum urut : “<< endl;

for (i=0; i<30; i++)
```

```
{
values[i]=rand()%100;
cout << values[i] <<" ";
}
cout << endl;

bubble_sort(values,30);

//ll
cout << "data yang sudah diurutkan dengan bubble sort : "<<endl;
for (i=0; i<30; i++)
cout << values[i] <<" ";
getche();
}
```

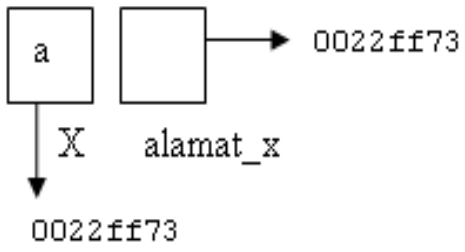
Latihan

- Tuliskan 10 digit nim anda masing-masing dan buatlah program sorting menggunakan 2 metode boleh dibuat ascending atau discending.

BAB IV
POINTER

Pointer adalah suatu variabel penunjuk, berisi nilai yang menunjuk alamat suatu lokasi memori tertentu. Jadi pointer **tidak** berisi nilai data, melainkan berisi suatu alamat memori atau null jika tidak berisi data. Pointer yang tidak diinisialisasi disebut **dangling pointer.** Lokasi memori tersebut bisa diwakili sebuah **variabel** atau dapat juga berupa nilai alamat memori **secara langsung**.[14]

- Kita memiliki variabel **X** yang berisi nilai karakter '**A**'
- Oleh kompiler, nilai '**A**' ini akan disimpan di suatu alamat tertentu di memori.
- Alamat variabel **X** dapat diakses dengan menggunakan statemen **&X**.
- Jika kita ingin menyimpan alamat dari variabel **X** ini, kita dapat menggunakan suatu variabel.
- misalnya **char Y = &X;**
- **Y** adalah suatu variabel yang berisi alamat dimana nilai **X**, yaitu '**A**' disimpan.
- Variabel **Y** disebut variabel pointer atau sering disebut **POINTER** saja.



Program Pointer :

```
int main()
{
   char *x;
   char y = 'a';
   x = &y;
```

[14] Hariyanto, Bambang,2000,Struktur Data,Bandung

```
printf("Nilai y = %c disimpan dalam alamat %p atau %x dalam Hexadecimal", y, x, x);
   return 0;
}
```

A. Variabel Pointer

Variabel pointer dapat dideklarasikan dengan tipe data apapun. Pendeklarasian variabel pointer dengan tipe data tertentu digunakan untuk menyimpan alamat memori yang berisi data sesuai dengan tipe data yang dideklarasikan, **bukan** untuk berisi nilai bertipe data tertentu. Tipe data digunakan sebagai lebar data untuk alokasi memori (misal char berarti lebar datanya 1 byte, dst). Jika suatu variabel pointer dideklarasikan bertipe float, berarti variabel pointer tersebut **hanya bisa** digunakan untuk menunjuk alamat memori yang berisi nilai bertipe float juga.

B. Operator Pointer

1. Operator Dereference (&)

Operator Dereference (&) yaitu operator yang berfungsi mendeklarasikan suatu variabel didalam penggantian memori. operator ini biasa disebut dengan “address of”. Jika ingin mengetahui dimana variable akan disimpan, kita dapat memberikan tanda ampersand (&) didepan variable , yang berarti“address of”.

Dari contoh program diatas, nilai a menjadi angka 0096F7DC, dengan menambahkan "&" pada variabel panjang, inilah yang dimaksud operator dereference, yaitu mengubah suatu nilai variabel menjadi alamat memori.

2. Operator Reference (*)

Operator Reference ini berkebalikan dengan operator Dereference dimana nilai variabel menjadi alamat memory sedangkan operator reference mengubah alamat memory menjadi nilai variabel.

Operator Reference (*) yaitu operator yang dapat mengakses secara langsung nilai yang terdapat didalam variabel yang berpointer, hal ini dapat dilakukan dengan menambahkan identifier asterisk (*). Operator ini biasa disebut dengan“value pointed by”

Dari contoh program diatas, nilai *a menjadi angka 20, dengan menambahkan "*" pada variabel a, dimana alamat memory 0096F7DC kembali menjadi nilai variabel yaitu 20, inilah yang dimaksud operator reference, yaitu mengubah suatu alamat memori kembali menjadi nilai variabel.

Operator *	Mendapatkan nilai data dari variabel pointer	Contoh:
		`int *alamat;`
		`int nilai = 10;`
		`alamat = &nilai;`
		`printf("%d",*alamat);`
			Hasil:
			10

Operator &	Mendapatkan alamat memori dari variabel pointer	Contoh:
		`int *alamat;`
		`int nilai = 10;`
		`alamat = &nilai;`
		`printf("%p",alamat);`
			Hasil:
			22FF70

C. OPERASI POINTER

Operasi Pointer terbagi menjadi beberapa operasi

1. **Operasi assignment**

Antar variabel pointer dapat dilakukan operasi assignment.

Contoh 1: Assignment dan sebuah alamat dapat ditunjuk oleh lebih dari satu pointer

Contoh 2: Mengisi variabel dengan nilai yang ditunjuk oleh sebuah variabel pointer

Contoh 3: Mengoperasikan isi variabel dengan menyebut alamatnya dengan pointer

Contoh 4: Mengisi dan mengganti variabel yang ditunjuk oleh pointer

2. Operasi aritmatika

Pada pointer dapat dilakukan operasi aritmatika yang akan menunjuk suatu alamat memori baru. Hanya nilai **integer** saja yang bisa dioperasikan pada variabel pointer. Biasanya hanya operasi **penambahan/pengurangan** saja.

Misal pointer X bertipe int (2 bytes), maka X+1 akan menunjuk pada alamat memori sekarang (mis. 1000) ditambah sizeof(X), yaitu 2, jadi 1002.

Program

```
int main()
{
   float y, *x1, *x2;
   y = 12.34;
   x1 = &y;
   x2 = x1;
   printf("Nilai y yang ditunjukkan oleh x1 adalah %2.2f di alamat %p\n", y, &y);
   printf("Nilai y yang ditunjukkan oleh x1 adalah %2.2f di alamat %p\n", *x2, x2);
   return 0;
}
```

%f untuk Float

%2.2f untuk Float 2 angka sebelum koma dan 2 angka setelah koma

BAB V
FUNCTION

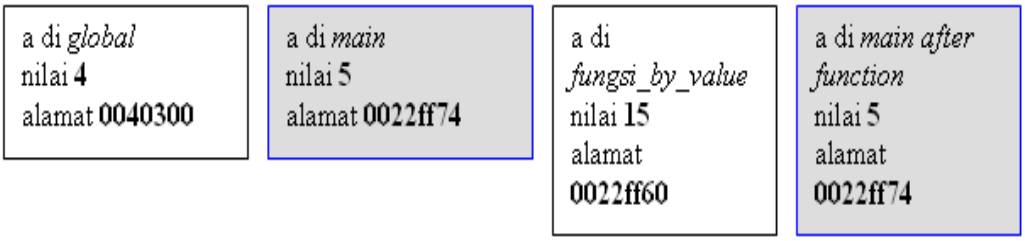
Fungsi/function adalah bagian dari program yang memiliki **nama** tertentu yang unik, digunakan untuk mengerjakan suatu pekerjaan tertentu, serta letaknya **dipisahkan** dari bagian program yang menggunakan/memanggil fungsi tersebut. [15]

- Dapat melakukan pendekatan *top-down* dan *divide-and-conquer*: program besar dapat dipisah menjadi program-program kecil.
- Dapat dikerjakan oleh beberapa orang sehingga koordinasi mudah.
- Kemudahan dalam mencari kesalahan-kesalahan karena alur logika jelas dan kesalahan dapat dilokalisasi dalam suatu modul tertentu saja.
- Modifikasi program dapat dilakukan pada suatu modul tertentu saja tanpa mengganggu program keseluruhan.
- Mempermudah dokumentasi.
- *Reusability*: Suatu fungsi dapat digunakan kembali oleh program atau fungsi lain
- Yang dikirimkan ke fungsi adalah **nilainya**, *bukan* **alamat memori** letak dari datanya
- Fungsi yang menerima kiriman nilai ini akan menyimpan nilainya di **alamat terpisah** dari nilai asli yang digunakan oleh program yang memanggil fungsi tersebut
- Karena itulah pengubahan nilai di dalam fungsi **tidak akan berpengaruh** pada nilai asli di program yang memanggil fungsi walaupun keduanya menggunakan nama variabel yang sama
- Sifat pengiriman **satu arah**, dari program pemanggil ke fungsi yang dipanggil saja.
- Parameter bisa berupa ungkapan (statemen)
  Lihat Contoh!

[15] Hariyanto, Bambang,2000,Struktur Data,Bandung

```c
#include <stdio.h>
#include <conio.h>

int a=4;

void getAGlobal(){
    printf("A Global adalah %d alamatnya %p\n",a,&a);
}

void fungsi_by_value(int a){
    a = a * 3;
    printf("A by value adalah = %d alamatnya adalah %p\n",a,&a);
}

int main(){
    int a = 5;
    getAGlobal();
    printf("A main adalah = %d alamatnya adalah %p\n",a,&a);
    fungsi_by_value(a);
printf("A main setelah fungsi dipanggil adalah = %d alamatnya adalah %p\n",a,&a);
    getch();
    return 0;
}
```

```
F:\Documents and Settings\Administrator\My Documents\coba2.exe
A Global adalah 4 alamatnya 00403000
A main adalah = 5 alamatnya adalah 0022FF74
A by value adalah = 15 alamatnya adalah 0022FF60
A main setelah fungsi dipanggil adalah = 5 alamatnya adalah 0022FF74
```



Yang dikirimkan adalah **alamat memori** letak dari nilai datanya, *bukan* **nilai datanya**. Fungsi yang menerima parameter ini akan menggunakan/mengakses data dengan **alamat yang sama** dengan alamat nilai datanya. Karena itulah pengubahan nilai di fungsi **akan mengubah** juga nilai asli di program pemanggil fungsi tersebut. Pengiriman parameter by reference adalah pengiriman **dua arah**, yaitu dari program pemanggil ke fungsi dan sebaliknya dari fungsi ke program pemanggilnya. Pengiriman parameter by reference **tidak dapat** digunakan untuk suatu ungkapan (statemen), hanya bisa untuk variabel, konstanta atau elemen array saja.

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

int a=4;

void getAGlobal(){
     printf("A Global adalah %d alamatnya %p\n",a,&a);
}

void fungsi_by_ref(int *a){
     *a = *a * 3;
     printf("A by ref adalah = %d alamatnya adalah %p\n",*a,a);
}

void main(){
     int a = 5;
     getAGlobal();
     printf("A main adalah = %d alamatnya adalah %p\n",a,&a);
     fungsi_by_ref(&a);
     printf("A main setelah fungsi dipanggil adalah = %d alamatnya
adalah %p\n",a,&a);
     getch();
}
```

Menggunakan asteris/bintang (*)

Menggunakan asteris/bintang (*)

Pengiriman parameter berupa array sebenarnya adalah pengiriman **by reference**, yang dikirimkan adalah **alamat elemen pertama** dari array, bukan seluruh nilai-nilai arraynya. Pada parameter formal, alamat elemen pertama dari array dapat ditulis berupa nama array saja tanpa ditulis indeksnya (kosong). Pada parameter aktual, penulisan dilakukan dengan menuliskan nama arraynya saja

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

void isi(int data[]){
     for(int i=0;i<5;i++)
        data[i] = i+1;
}

void tampil(int data[]){
     for(int i=0;i<5;i++)
        printf("%d ",data[i]);
}

int main(){
     int mydata[5];
     printf("pengisian...\n");
     isi(mydata);
     printf("selesai.\n");
     printf("tampilkan...\n");
     tampil(mydata);
     getch();
}
```

BAB VI
STACK

STACK (tumpukan) adalah list linier yang:

1. dikenali elemen puncaknya (TOP)
2. aturan penyisipan dan penghapusan elemennya tertentu

   Penyisipan selalu dilakukan "di atas" TOP

   Penghapusan selalu dilakukan pada TOP

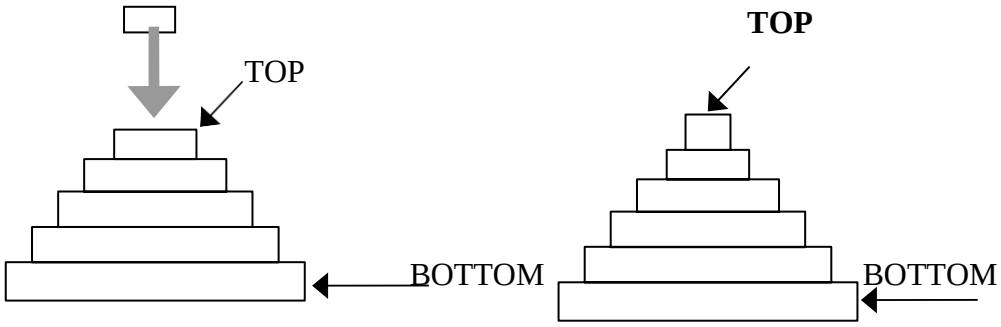
Karena aturan penyisipan dan penghapusan semacam itu, TOP adalah satu-satunya alamat tempat terjadi operasi, elemen yang ditambahkan paling akhir akan menjadi elemen yang akan dihapus Dikatakan bahwa elemen Stack akan tersusun secara LIFO *(Last In First Out).*[16]

Struktur data ini banyak dipakai dalam informatika, misalnya untuk merepresentasi :

- pemanggilan prosedur,
- perhitungan ekspresi artimatika,
- rekursifitas,
- *backtracking*,

dan algoritma lanjut yang lain.

[16] Sjukani, Moh,2012,*Struktur Data (Algoritma dan Struktur Data dengan C,C++)*,Jakarta:Mitra Wacana Media.



Perhatikan bahwa dengan definisi semacam ini, representasi tabel sangat tepat untuk mewakili Stack, karena operasi penambahan dan pengurangan hanya dilakukan di salah satu “ujung” tabel. “Perilaku” yang diubah adalah bahwa dalam Stack, operasi penambahan hanya dapat dilakukan di salah satu ujung. Sedangkan pada tabel, boleh di mana pun.

A. Definisi Fungsional

Jika diberikan S adalah Stack dengan elemen ElmtS, maka definisi fungsional stack adalah:

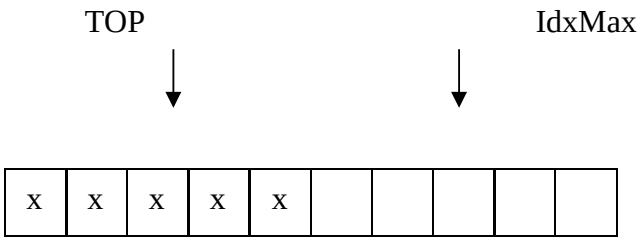
CreateEmpty : □	S	{ Membuat sebuah stack kosong}
IsEmpty	: S □ boolean	{ Test stack kosong, true jika stack kosong, false jika S tidak kosong }
IsFull	: S □ boolean	{ Test stack penuh, true jika stack penuh, false jika S tidak }
Push	: ElmtS x S □ S	{ Menambahkan sebuah elemen ElmtSsebagai TOP, TOP berubah nilainya }
Pop	: S □ S x ElmtS	{ Mengambil nilai elemen TOP, sehingga TOP yang baru adalah elemen yang datang sebelum elemen TOP, mungkin S menjadi kosong }

Definisi selektor adalah:
Jika S adalah sebuah Stack, maka Top(S) adalah alamat elemen TOP, di mana operasi penyisipan/penghapusan dilakukan, InfoTop(S) adalah informasi yang disimpan pada Top(S). Definisi Stack kosong adalah Stack dengan Top(S)=Nil (tidak terdefinisi).

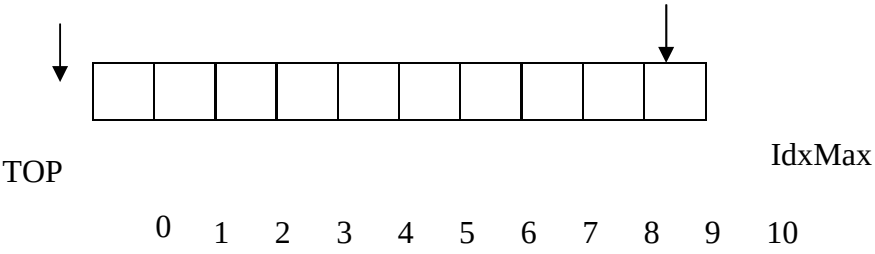
B. Implementasi Stack dengan Tabel

Tabel dengan hanya representasi TOP adalah indeks elemen Top dari Stack. Jika Stack kosong, maka TOP=0.

Ilustrasi Stack tidak kosong, dengan 5 elemen :



Ilustrasi Stack kosong:



C. Operasi Stack

Ada empat operasi dasar yang didefinisikan pada stack, yaitu :

1. CREATE(stack)
2. ISEMPTY(stack)
3. PUSH(elemen,stack)
4. POP(stack)

1. CREATE

Operator ini berfungsi untuk membuat sebuah stack kosong dan didefinisikan bahwa :

NOEL(CREATE(S)) = 0 dan TOP(CREATE(S)) = null

2. ISEMPTY

Operator ini berfungsi untuk menentukan apakah suatu stack adalah stack kosong. Operasinya akan bernilai boolean, dengan definisi sebagai berikut :

ISEMPTY(S) = true, jika S adalah stack kosong
= false, jika S bukan stack kosong

atau

ISEMPTY(S) = true, jika NOEL(S) = 0
= false, jika NOEL(S) ≠ 0

Catatan : ISEMPTY(CREATE(S)) = true.

3. PUSH

Operator ini berfungsi untuk menambahkan satu elemen ke dalam stack. Notasi yang digunakan adalah :

PUSH(E,S)

Artinya : menambahkan elemen E ke dalam stack S.

Elemen yang baru masuk ini akan menempati posisi TOP.

Jadi : **TOP(PUSH(E,S)) = E.**

Akibat dari operasi ini jumlah elemen dalam stack akan bertambah, artinya NOEL(S) menjadi lebih besar atau stack menjadi tidak kosong (ISEMPTY(PUSH(E,S)) = false).

4. POP

Operator ini berfungsi untuk mengeluarkan satu elemen dari dalam stack. Notasinya :

POP(S)

Elemen yang keluar dari dalam stack adalah elemen yang berada pada posisi TOP. Akibat dari operasi ini jumlah elemen stack akan berkurang atau NOEL(S) berkurang dan elemen pada posisi TOP akan berubah. Operator POP ini tidak dapat digunakan pada stack kosong, artinya :

POP(CREATE(S)) = error condition

Catatan : **TOP(PUSH(E,S)) = E**

D. DEKLARASI STACK PADA BAHASA PEMROGRAMAN

Dalam bahasa pemrograman, untuk menempatkan stack biasanya digunakan sebuah array. Tetapi perlu diingat di sini bahwa stack dan array adalah dua hal yang berbeda. Misalkan suatu variabel S adalah sebuah stack dengan 100 elemen. Diasumsikan elemen S adalah integer dan jumlah elemennya maksimum adalah 100 elemen. Untuk mendeklarasikan stack dengan menggunakan array, harus dideklarasikan pula variabel lain yaitu TOP_PTR yang merupakan indeks dari array. Variabel TOP_PTR ini digunakan untuk menyatakan elemen yang

berada pada posisi TOP dalam stack tersebut. Selanjutnya gabungan kedua variabel ini diberi nama STACK_STRUCT.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#define maxstack 200
#include <conio.h>

using namespace std;

struct stack
{
   int atas;
   char data[maxstack];
};

char dta[maxstack];
struct stack stackbaru;

/*--Class inisialisasi--*/
void inisialisasi()
{
   stackbaru.atas=-1;
}

/*--Class untuk mengecek kepenuhan data--*/
bool ispenuh()
{
   if (stackbaru.atas == maxstack-1)return true;
   else return false;
}
```

```
/*--Class utnuk mengecek kekosongan data--*/
bool iskosong()
{
   if (stackbaru.atas == -1) return true;
   else return false;
}

/*Class untuk mengisi stack (menyiapkan data)--*/
void push(char dta)
{
   if (ispenuh() == false)
   {
      stackbaru.atas++;
      stackbaru.data[stackbaru.atas]=dta;
   }
   else
   {
      puts("\nMaaf Stack Penuh");
   }
}

/*--Class untuk mengambil isi stack--*/
void pop()
{
   while (iskosong() == false)
   {
      cout<<stackbaru.data[stackbaru.atas];
      stackbaru.atas--;
   }
}

/*--Class untuk mencetak stack--*/
```

```
void print()
{
   cout<<"";
   for(int i=0; i<=stackbaru.atas; i++)
   {
      cout<<stackbaru.data[i];
   }
}

/*--Class untuk membersihkan layar--*/
void clear()
{
   stackbaru.atas = -1;
}

/*--Main Program--*/
int main()
{
   char kata[200];

   cout<<"---PROGRAM---"<<endl;
   cout<<"---PEMBALIK---"<<endl;
   cout<<"---KATA---"<<endl;
   cout<<"Masukan kalimat : \n"<<endl;

   gets(kata);

   for(int i=0; kata[i]; i++)
      push(kata[i]);
   cout<<"---------------------------------\n\n"<<endl;

print();
```

```
cout<<"  Menjadi  ";
pop();
cout<"\n\n\n";
getche();
}
```

E. NOTASI POSTFIX

Bentuk aplikasi stack yang lain adalah mengubah suatu ekspresi aritmatik (string) ke dalam notasi postfix. Notasi postfix ini digunakan oleh compiler untuk menyatakan suatu ekspresi aritmatik dalam bahasa tingkat tinggi (high level language). Stack digunakan oleh compiler untuk mentransformasikan ekspresi aritmatik menjadi suatu ekspresi dalam bentuk/notasi postfix.[17]

Contoh :

Misal diberikan ekspresi aritmatik : A + B ;

Maka bentuknya dalam notasi postfix menjadi : AB+

Urutan (prioritas) dari operator adalah :

1. Perpangkatan (^)
2. Perkalian (*) atau Pembagian (/)
3. Penjumlahan (+) atau Pengurangan (-)

Aturan yang digunakan dalam proses transformasi tersebut adalah :

1. Ekspresi aritmatik yang diberikan di- "Scan" dari kiri ke kanan.
2. Bila simbol yang di-scan adalah "(", maka simbol tersebut di push ke dalam stack.
3. Bila simbol yang di-scan adalah ")", maka seluruh isi stack di pop keluar mulai dari simbol "(" yang pertama ditemukan dalam stack.
4. Bila simbol adalah operator, maka dilakukan perbandingan dulu dengan simbol (operator) yang berada pada posisi top dalam stack.

[17]Sjukani, Moh,2012,*Struktur Data (Algoritma dan Struktur Data dengan C,C++)*,Jakarta:Mitra Wacana Media

a. Jika derajatnya setara atau lebih rendah dari simbol yang berada pada posisi top, maka top stack di-pop keluar sebagai output dan simbol yang baru di-push ke dalam stack.
b. Jika derajatnya lebih tinggi dari simbol yang berada pada posisi top, maka simbol (operator) yang di-scan tersebut di-push ke dalam stack.

5. Bila simbol yang di-scan adalah operand, maka simbol tersebut langsung sebagai output.
6. Bila simbol adalah ";" maka seluruh isi stack di-pop sebagai output.

Contoh :

Misal diberikan sebuah ekspresi aritmatik dengan bentuk sbb:

((A + B) * C / D + E ^ F) / G ;

Selanjutnya akan dicari bentuk ekspresi diatas dalam notasi postfix.

Proses yang dilakukan dapat digambarkan dalam tabel berikut :

Urutan Proses	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Simbol Yang di Scan	(	(	A	+	B	)	*	C	/	D	+	E	^	F	)	/	G	;
Top	(	((	((	+ ((	+ ((	(	* (	* (	/ (	/ (	+ (	+ (	^ + (	^ + (		/	/	
Output			A		B	+		C	*	D	/	E		F	^+		G	/

Jadi ekspresi aritmatik : ((A + B) * C / D + E^F) / G ;

dalam notasi postfix menjadi : AB+D*C/EF^+G/

BAB VII
QUEUE

Queue (antrian) adalah struktur data yang meniru antrian orang yang sedang menunggu pelayanan misalnya di depan seorang teller bank, atau antrian orang yang sedang membeli karcis pertunjukan. Apabila diperhatikan dengan seksama maka penambahan orang pada suatu antrian selalu dilakukan pada urutan paling belakang (*rear of queue*), dan pelayanan selalu dilakukan pada urutan depan (*front of queue*) sehingga urutan proses antrian sering disebut FIFO(*First In First Out*). Yang pertama masuk antrian, itulah yang pertama dilayani.[18]

A. Pendeklarasian queue

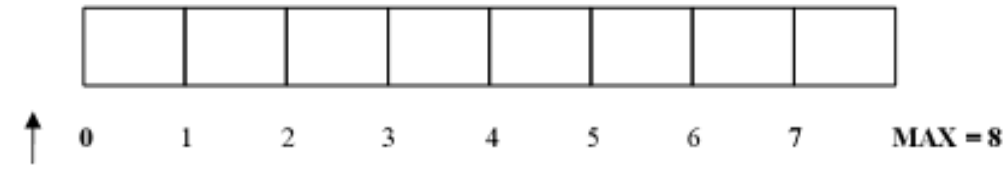
```
//deklarasi queue dengan struct dan array
struct QUEUE
{
  int data[5];
  int head;
  int tail;
};

//deklarasi variabel antrian dari struct
QUEUE antrian;
```

B. Operasi Queue

1. Create()

- Untuk menciptakan dan menginisialisasi Queue
- Dengan cara membuat Head dan Tail = -1

[18] Dwi, Sanjaya,2005,Asyiknya Belajar Struktur Data di Planet C++,Jakarta:Elex Media Komputindo,Jakarta



Head = -1

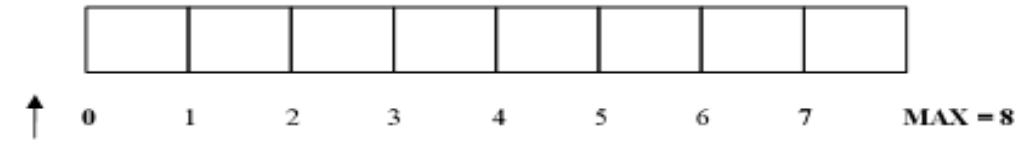
Antrian pertama kali

Tail = -1

```
void Create(){
     antrian.head=antrian.tail=-1;
}
```

2. IsEmpty()

- Untuk memeriksa apakah Antrian sudah penuh atau belum
- Dengan cara memeriksa nilai ***Tail,*** jika Tail = -1 maka ***empty***
- Kita tidak memeriksa ***Head,*** karena ***Head*** adalah tanda untuk kepala antrian (elemen pertama dalam antrian) yang tidak akan berubah-ubah
- Pergerakan pada Antrian terjadi dengan penambahan elemen Antrian kebelakang, yaitu menggunakan nilai ***Tail***



Head = -1

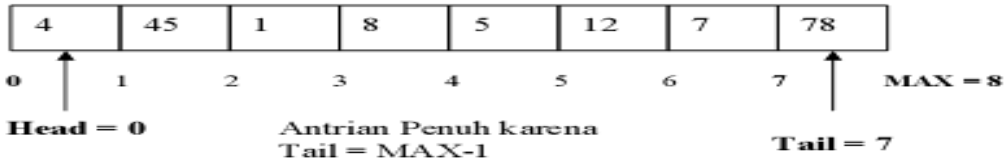
Antrian Kosong
karena Tail = -1

Tail = -1

```
int IsEmpty(){
     if(antrian.tail==-1)
          return 1;
     else
          return 0;
}
```

3. Fungis IsFull

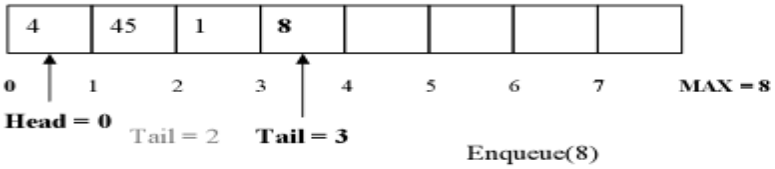
- Untuk mengecek apakah Antrian sudah penuh atau belum
- Dengan cara mengecek nilai Tail, jika Tail >= MAX-1 (karena MAX-1 adalah batas elemen array pada C) berarti sudah penuh




```
int IsFull(){
      if(antrian.tail==MAX-1) return 1;
      else return 0;
}
```

4. Enqueue

- Untuk menambahkan elemen ke dalam Antrian, penambahan elemen selalu ditambahkan di elemen paling **belakang**
- Penambahan elemen selalu menggerakan variabel Tail dengan cara increment counter Tail terlebih dahulu




```
void Enqueue(int data){
     if(IsEmpty()==1){
          antrian.head=antrian.tail=0;
          antrian.data[antrian.tail]=data;
          printf("%d masuk!",antrian.data[antrian.tail]);
     } else
     if(IsFull()==0){
          antrian.tail++;
          antrian.data[antrian.tail]=data;
          printf("%d masuk!",antrian.data[antrian.tail]);
     }
}
```

5. Dequeue()

- Digunakan untuk menghapus elemen terdepan/pertama (head) dari Antrian
- Dengan cara menggeser semua elemen antrian kedepan dan mengurangi Tail dgn 1
- Penggeseran dilakukan dengan menggunakan looping

BAB VIII
LINKED LIST

Salah satu bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis adalah senarai berkait (*linked list*).Suatu senarai berkait (*linked list*) adalah suatu simpul (*node*) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. Suatu simpul dapat berbentuk suatu struktur atau *class*. Simpul harus mempunyai satu atau lebih elemen struktur atau class yang berisi data.
Secara teori, *linked list* adalah sejumlah *node* yang dihubungkan secara linier dengan bantuan *pointer*. Dikatakan single linked apabila hanya ada satu pointer yang menghubungkan setiap *node* single.

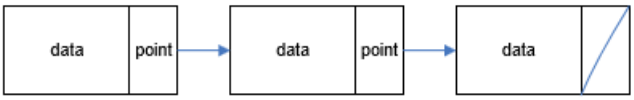
Senarai berkait adalah struktur data yang paling dasar. Senarai berkait terdiri atas sejumlah unsur-unsur dikelompokkan, atau terhubung, bersama-sama di suatu deret yang spesifik. Senarai berkait bermanfaat di dalam memelihara koleksi-koleksi data, yang serupa dengan array/larik yang sering digunakan. Bagaimanapun juga, senarai berkait memberikan keuntungan-keuntungan penting yang melebihi array/larik dalam banyak hal. Secara rinci, senarai berkait lebih efisien di dalam melaksanakan penyisipan-penyisipan dan penghapusan-penghapusan. Senarai berkait juga menggunakan alokasi penyimpanan secara dinamis, yang merupakan penyimpanan yang dialokasikan pada runtime . Karena di dalam banyak aplikasi, ukuran dari data itu tidak diketahui pada saat kompile, hal ini bisa merupakan suatu atribut yang baik juga. Setiap node akan berbentuk struct dan memiliki satu buah field bertipe struct yang sama.[19]

Deklarasi node:

```
struct node
{
char nama[20]
; int umur;
float tinggi;
```

[19] Hariyanto, Bambang,2000,Struktur Data,Bandung

*node *next; // Pointer menyambung ke node selanjutnya*
};



Gambar sebuah node

- Bagian pertama, disebut medan informasi, berisi informasi yang akan disimpan dan diolah.
- Bagian kedua, disebut medan penyambung (link field), berisi alamat simpul berikutnya

Pada gambar di atas, pointer awal menunjuk ke simpul pertama dari senerai tersebut. Medan penyambung (pointer) dari suatu simpul yang tidak menunjuk simpul lain disebut **pointer kosong**, yang nilainya dinyatakan sebagai **null** (**null** adalah kata baku yang berarti bahwa pointer 0 atau bilangan negatif). Jadi kita bisa melihat bahwa dengan hanya sebuah pointer Awal saja maka kita bisa membaca semua informasi yang tersimpan dalam senarai.

Operasi pada Linked list yaitu :

1. Menambah Simpul
2. Menghapus simpul
3. Mencari Informasi pada suatu Linked list
4. Mencetak Simpul

8.1 SINGLE LINKED LIST

Single Linked list adalah Daftar terhubung yang setiap simpul pembentuknya mempunyai satu rantai(link) ke simpul lainnya. Pembentukan linked list tunggal memerlukan :

1. deklarasi tipe simpul

2. deklarasi variabel pointer penunjuk awal Linked list

3. pembentukan simpul baru

4. pengaitan simpul baru ke Linked list yang telah terbentuk

Ada beberapa operasi yang dapat dibuat pada senarai tersebut, diantaranya: tambah, hapus dan edit dari senarai tersebut.

A. Menambah Sampul Di Belakang Single Linked List

Sekarang kita akan mempelajari bagaimana menambah simpul baru ke dalam senerai berantai. Kita anggap bahwa simpul baru yang akan ditambah selalu menempati posisi setelah posisi yang terakhir dari senerai berantai yang diketahui. Untuk menjelaskan operasi ini baiklah kita gunakan deklarasi pointer dan simpul seperti di bawah ini:
Program :

```
struct node
{
char nama[20];
int umur;
float tinggi;
node *next;
};
```

B. MENAMBAHKAN SIMPUL DIAKHIR SINGLE LINKED LIST

Misalkan simpul baru yang dibentuk diberi nama temp. Untuk menambahkan simpul baru perlu kita uji apakah Linked list masih kosong atau tidak. Linked list yang kosong ditandai dengan awal_ptr bernilai NULL. Jika tidak kosong maka dibaca senarai yang ada mulai dari posisi awal sampai terakhir misalkan dengan menggunakan pointer temp2. Simpul terakhir ditandai dengan medan penyambung dari temp2 bernilai NULL. Jika sudah berada pada simpul terakhir kita tunjukkan medan penyambung temp2 dengan temp.
Program lengkap menambahkan simpul diakhit senarai sebagai berikut:

```
node *temp, *temp2; // pointer sementara
// Isi data temp = new node; //menciptakan node baru
cout << "Nama : ";cin >> temp->nama;
cout << "Umur : ";cin >> temp->umur;
cout << "Tinggi : ";cin >> temp->tinggi;
temp->next = NULL;
// Inisialisasi pointer ketika masih kosong
if (awal_ptr == NULL)
{
awal_ptr = temp;
posisi = awal_ptr;
}
else
{
temp2 = awal_ptr; // list tidak kosong
while (temp2->next != NULL)
{
temp2 = temp2->next;
// pindah ke link berikutnya
}
temp2->next = temp;
}
```

Menghapus simpul diakhir list kita gunakan cara sebagai berikut :
Untuk menghapus simpul diakhir perlu kita uji apakah senarai masih kosong atau tidak, jika kosong tampilkan pesan bahwa list masih kosong. Jika tidak kosong perlu diuji apakah jumlah simpul dalam senarai hanya satu yang ditandai dengan medan penyambung yang bernilai null. Jika simpul lebih dari satu maka dibaca semua simpul sampai simpul terakhir yaitu medan penyambung bernilai null.

```
node *temp1, *temp2;
if (awal_ptr == NULL)
```

```
cout << "List kosong!" << endl;
else
{
temp1 = awal_ptr;
if (temp1->next == NULL)
{
delete temp1
; awal_ptr = NULL;
}
else
{
while (temp1->next != NULL)
{
temp2 = temp1;
temp1 = temp1->next;
} delete temp1; temp2->next = NULL; }
}
```

C. MENGHAPUS SIMPUL DIAWAL SINGLE LINKED LIST

Menghapus simpul diawal dilakukan dengan cara menampung simpul awal pada pointer temp, kemudian mengarahkan awal_ptr ke simpul selanjutnya. Kemudian simpul yang ditampung dalam pointer temp dihapus.

```
node *temp;
temp = awal_ptr;
awal_ptr = awal_ptr->next;
delete temp;
```

D. MENAMPILKAN SINGLE LINKED LIST

Sebelum menampilkan linked list perlu kita uji apakah senarai kosong atau tidak. Jika senarai kosong kita tampilkan pesan bahwa List kosong. Jika senarai tidak kosong maka kita baca senarai mulai posisi awal sampai simpul

terakhir.

```
node *temp;
temp = awal_ptr;
cout << endl;
if (temp == NULL)
cout << "List kosong!" << endl;
else
{
while (temp != NULL)
{
// Menampilkan isi cout << "Nama : " << temp->nama << " ";
cout << "Umur : " << temp->umur << " ";
cout << "Tinggi : " << temp->tinggi;
if (temp == posisi)
cout << " <– Posisi node";
cout << endl; temp = temp->next;
}
cout << "Akhir list!" << endl;
}
```

Program Lengkap:

```
#include <iostream.h>
struct node {
char nama[20];
int umur;
float tinggi;
node *next; };
node *awal_ptr=NULL;
node *posisi;
int pilih;
```

```
 void tambah_simpul_akhir()
 {
node *temp, *temp2; //simpul sementara
 //isi data
 temp=new node;
 cout<<"Nama   : ";cin>>temp->nama;
 cout<<"Umur   : ";cin>>temp->umur;
 cout<<"Tinggi : ";cin>>temp->tinggi;
 temp->next=NULL;
 //inisialisasi pointer ketika kosong
   if(awal_ptr==NULL)
   {
   awal_ptr=temp;
   posisi=awal_ptr;
   }
  else
   {
   temp2=awal_ptr;
   while(temp2->next != NULL)
    {
       temp2 = temp2->next;
    }
    temp2->next=temp;
 }  }

void tampil_senarai()
{
 node *temp;
 temp = awal_ptr;
 if(temp == NULL)
    cout<<"List kosong"<<endl;
 else
```

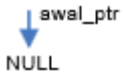
```
 {
   cout<<endl<<endl;
   while(temp != NULL)
   {
      cout<<"Nama : "<<temp->nama<<"  ";
      cout<<"Umur : "<<temp->umur<<"  ";
      cout<<"Tinggi : "<<temp->tinggi;
      if (temp == posisi)
         cout<<"   << posisi simpul";
      cout<<endl;
      temp=temp->next;
   }
   cout<<"Akhir list"<<endl;
 } }
void hapus_simpul_akhir()
{
 node *temp1, *temp2;
 if (awal_ptr == NULL)
 cout << "List kosong!" << endl;
 else
 {
 temp1 = awal_ptr;
  if (temp1->next == NULL)
  {
  delete temp1;
  awal_ptr = NULL;
  }
  else
  {
  while (temp1->next != NULL)
  {
   temp2 = temp1;
```

```
    temp1 = temp1->next;
    }
    delete temp1;
    temp2->next = NULL;
     }   }
}
void hapus_simpul_awal()
{
 node *temp;
 temp = awal_ptr;
 awal_ptr = awal_ptr->next;
 delete temp;
}
void main()
{   awal_ptr=NULL;
  do
  {
    tampil_senarai();
    cout<<"Menu Pilihan"<<endl;
    cout<<"0. Keluar program"<<endl;
    cout<<"1. Tambah simpul akhir"<<endl;
    cout<<"2. Hapus simpul akhir"<<endl;
    cout<<"3. Hapus simpul awal"<<endl;
     cout<<"Pilihan >> ";cin>>pilih;
     switch(pilih)
     {
        case 1: tambah_simpul_akhir();break;
        case 2: hapus_simpul_akhir();break;
        case 3: hapus_simpul_awal();break;
     }     }
 while(pilih !=0);  }
```

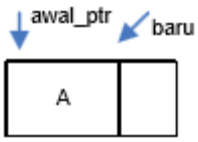
E. MENAMBAH SIMPUL DI DEPAN SINGLE LINKED LIST

Adalah menambahkan simpul baru yang dimasukkan diawal list. Proses penambahan simpul diawal list diilustrasikan sebagai berikut:

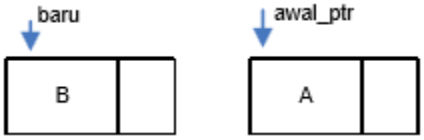
1. List masih kosong maka awal_ptr bernilai NULL (awal_ptr=NULL)



2. Masuk simpul baru, misalkan data A



3. Menambakan simpul diawal simpul A, misalkan simpul B Baru->next diisi dengan alamat dimana simpul data A berada, kemudian awal_ptr ditunjukkan ke simpul baru yang diciptakan.



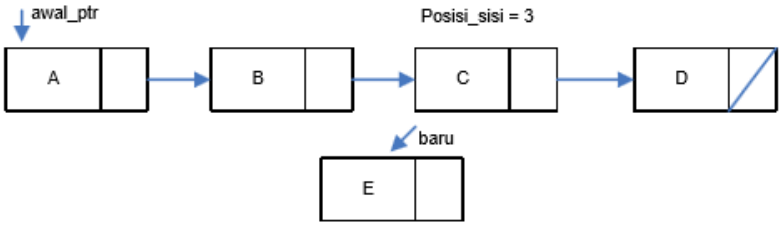
Program :

```
if(awal_ptr == NULL)
{
 awal_ptr=baru;
 awal_ptr->
 next = NULL;
}
else
 {
  baru->
 next = awal_ptr;
```

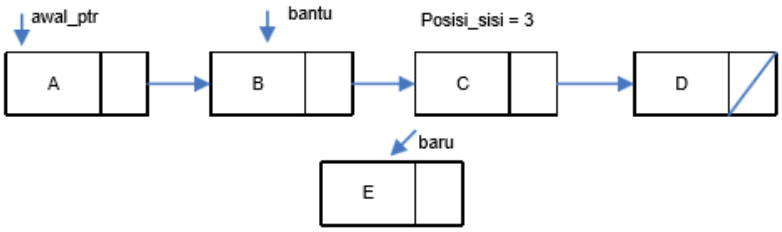
```
 awal_ptr = baru;
}
```

F. MENAMBAH SIMPUL DI TENGAH SINGLE LINKED LIST

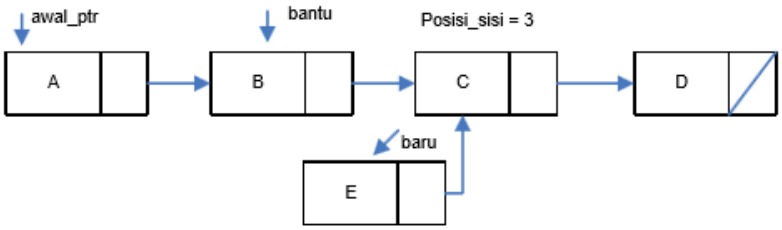
Proses penambahan di tengah berarti proses penyisipan data pada posisi tertentu. Oleh karena itu, posisi penyisipan sangat diperlukan. Ada beberapa kondisi yang harus diperhatikan ketika ingin melakukan penyisipandata, yaitu kondisi ketika linked list masih kosong, dan ketika linked list sudah mempunyai data. Proses penambahan data ketika linked list sudah mempunyai data. Ada 3 kondisi yang terjadi ketika akan melakukan proses penyisipan pada linked list yang sudah mempunyai data adalah :



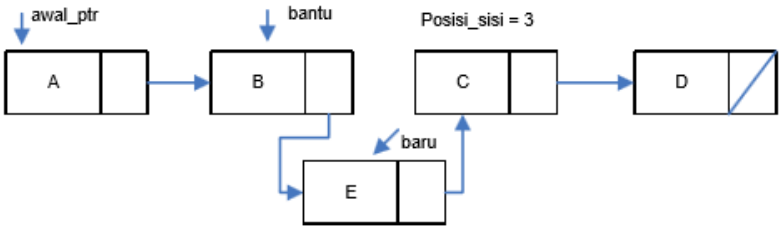
Langkah selanjutnya cari posisi elemen sebelum posisi sisip kemudian simpan dalam suatu variabel dengan nama bantu.



Kemudian sambungkan field next dari Baru ke posisi next dari bantu



Kemudian pindahkan field next dari bantu ke posisi data baru.



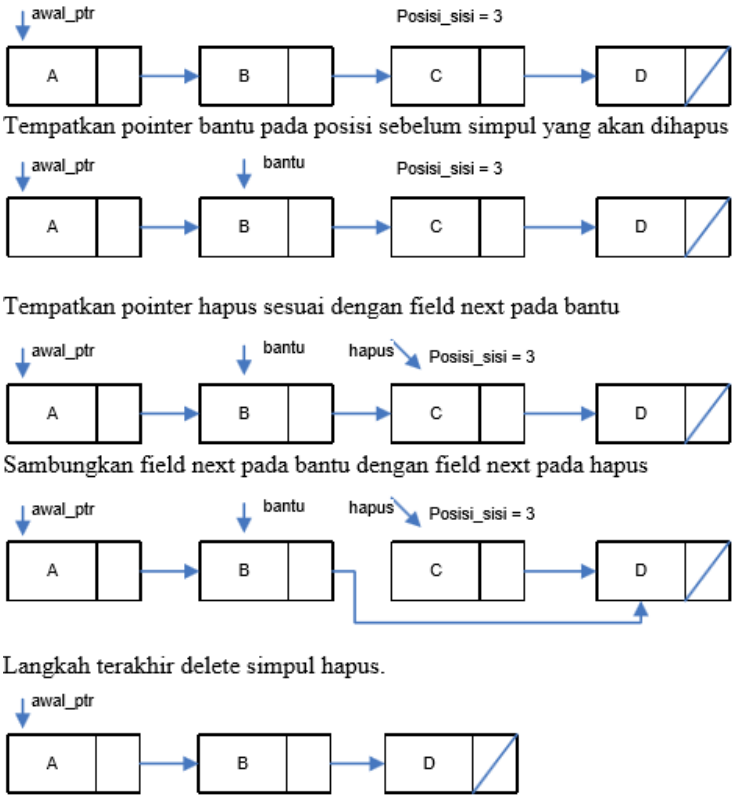
1. Posisi penyisipan di luar dari jangkauan linked list (posisi kurang dari 1 atau melebihi banyak data yang ada di linked list). Pada proses ini, jika terjadi posisi penyisipan kurang dari 1 maka proses yang dilakukan adalah proses penambahan data di awal, dan jika posisi diluar (>) dari banyak data maka proses yang dilakukan adalah proses penambahan data di akhir.
2. Posisi penyisipan di dalam jangkauan linked list. Contoh kalau ingin menyisipkan data pada posisi ke-3 (posisi_sisip=3).

Program :

```
 if(awal_ptr != NULL)
 {
 cout<<"Akan disisip setelah Data Ke ? : ";
 cin>>posisi_sisip;
 bantu=awal_ptr;
 baru =new node;
 for(int i=1;i<posisi_sisip-1;i++)
{
   if(bantu->next != NULL)
      bantu=bantu->next; else break;
 }
 cout << "Masukkan Nama : ";
 cin >> baru->nama;
 cout << "Masukkan Umur : ";
 cin >> baru->umur;
cout << "Masukkan tingggi : ";
 cin >> baru->tinggi; baru->
next=bantu->next;
 bantu->next=baru;
}
else
 {
  cout<<"Belum ada data !! silahkan isi data dulu....";
  getch();
 } }
```

G. MENGHAPUS SIMPUL DI TENGAH SINGLE LINKED LIST

Menghapus tengah sebuah simpul adalah menghilangkan simpul dari deret linked list. Misalkan kita akan menghapus simpul pada posisi 3



Program:

```
if(awal_ptr != NULL)
 {
  cout<<" Akan dihapus pada data ke : ";
  cin>>posisi_hapus;
  banyakdata=1;
  bantu=awal_ptr;
 while(bantu->next != NULL)
 {
   bantu=bantu->next;
   banyakdata++;
 }
 if((posisi_hapus<1)||(posisi_hapus>banyakdata))
 {
```

```
  cout<<"Belum ada data !! masukkan Data dula aja...\n";
 }
else
{
  bantu=awal_ptr;
  poshapus=1;
 while(poshapus<(posisi_hapus-1))
 {
  bantu=bantu->next;
  poshapus++;
 }
hapus=bantu->next;
bantu->
next=hapus->next;
delete hapus;
}
}
else
cout<<"Data Masih kosong, tidak bisa hapus data dari tengah! ";
getch();
}
```

Program lengkap:

```
#include <iostream.h>
#include <conio.h>
struct node
{
 char nama[20];
 int umur;
 float tinggi;
 node *next;
};
```

```
node *awal_ptr = NULL;
  node *posisi;         //digunakan untuk membaca sepanjang list
  int option = 0;

void tambah_awal_list()
{
  node *baru;
  baru = new node;
  cout << "Masukkan Nama     : ";
  cin >> baru->nama;
  cout << "Masukkan Umur     : ";
  cin >> baru->umur;
  cout << "Masukkan tingggi  : ";
  cin >> baru->tinggi;
  baru->next = NULL;
  if(awal_ptr == NULL)
  {
    awal_ptr=baru;
    awal_ptr->next = NULL;
  }
  else
  {
    baru->next = awal_ptr;
    awal_ptr = baru;
  }
}

void menambah_node_di_akhir()
  {
  node *temp, *temp2;   // Temporary pointers
```

```
  // menciptakan node baru
  temp = new node;
  cout << "Masukkan Nama     : ";
  cin >> temp->nama;  cout << "Masukkan Umur     : ";
  cin >>   temp->umur;  cout << "Masukkan tingggi  : ";
  cin >> temp->tinggi;  temp->next = NULL;

// Set up link pada node
  if (awal_ptr == NULL)
  {
    awal_ptr = temp;
    posisi = awal_ptr;
  }
  else
  {
    temp2 = awal_ptr;
    // node tidak NULL – list tidak kosong
    while (temp2->next != NULL)
    {
      temp2 = temp2->next;
      // Memindahkan pada next link dalam rantai
    }
  temp2->next = temp;
  }
}

void display_list()
{
  node *temp;
  temp = awal_ptr;
  cout << endl;
  if (temp == NULL)
```

```
   cout << "List kosong!" << endl;
  else
  {
    while (temp != NULL)
    {  // Menampilkan detail data
     cout << "nama : " << temp->nama << "  ";
     cout << "umur : " << temp->umur << "  ";
     cout << "tinggi : " << temp->tinggi;
     if (temp == posisi)
        cout << "     <<<< posisi node";
     cout << endl;
     temp = temp->next;
    }
    cout << "Akhir list!" << endl;
  }
}

void hapus_awal_node()
{
  node *temp;
  temp = awal_ptr;
  awal_ptr = awal_ptr->next;
  delete temp;
}

void hapus_akhir_node()
{
  node *temp1, *temp2;
  if (awal_ptr == NULL)
    cout << "List kosong!" << endl;
  else
  {
```

```
    temp1 = awal_ptr;
    if (temp1->next == NULL)
    {
      delete temp1;
      awal_ptr = NULL;
    }
    else
    {
      while (temp1->next != NULL)
      {
        temp2 = temp1;
        temp1 = temp1->next;
      }
      delete temp1;
      temp2->next = NULL;
    }

   }
}

void pindah_posisi_sebelumnya()
{
  if (posisi->next == NULL)
  cout << "Kamu berada pada akhir list." << endl;
  else
  posisi = posisi->next;
}

void pindah_posisi_berikutnya()
{
  if (posisi == awal_ptr)
    cout << "Kamu berada pada awal list" << endl;
```

```
  else
  {
    node *previous;     // deklarasi pointer
    previous = awal_ptr;
    while (previous->next != posisi)
    {
      previous = previous->next;
    }
    posisi = previous;
  }
}

void tambah_tengah_list()
{
  node *baru, *bantu;
  int posisi_sisip;
  if(awal_ptr != NULL)
  {
    cout<<"Akan disisip setelah Data Ke ? : ";
    cin>>posisi_sisip;
    bantu=awal_ptr;
    baru =new node;
    for(int i=1;i<posisi_sisip-1;i++) {
      if(bantu->next != NULL)
        bantu=bantu->next;
      else
        break;
    }
  cout << "Masukkan Nama     : ";
  cin >> baru->nama;
  cout << "Masukkan Umur     : ";
  cin >> baru->umur;
```

```
  cout << "Masukkan tingggi  : ";
  cin >> baru->tinggi;
  baru->next=bantu->next;
  bantu->next=baru;
  }
  else
  {
    cout<<"Belum ada data !! silahkan isi data dulu....";
    getch();
  }
}
void Hapus_tengah_list()
{
  int banyakdata,posisi_hapus,poshapus;
  node *hapus, *bantu;
  if(awal_ptr != NULL)
  {
    cout<<" Akan dihapus pada data ke : ";
    cin>>posisi_hapus;
    banyakdata=1;
    bantu=awal_ptr;
    while(bantu->next != NULL)
    {
      bantu=bantu->next;
      banyakdata++;
    }
    if((posisi_hapus<1)||(posisi_hapus>banyakdata))
    {
      cout<<"Belum ada data !! masukkan Data dula aja...\n";
    }
    else
    {
```

```
      bantu=awal_ptr;
      poshapus=1;
      while(poshapus<(posisi_hapus-1))
      {
        bantu=bantu->next;
        poshapus++;
      }
      hapus=bantu->next;
      bantu->next=hapus->next;
      delete hapus;
    }
  }
  else
    cout<<"Data Masih kosong, tidak bisa hapus data dari tengah! ";
  getch();
}


int main()
{
  awal_ptr = NULL;
  do
  {
    clrscr();
    display_list();
    cout << endl;
    cout << "MENU PILIHAN : " << endl;
    cout << "0. Keluar program." << endl;
    cout << "1. Tambah awal list." << endl;
    cout << "2. Tambah akhir list." << endl;
    cout << "3. Tambah tengah list."<< endl;
    cout << "4. Hapus awal list." << endl;
```

```
    cout << "5. Hapus akhir list." << endl;
    cout << "6. Hapus tengah list." << endl;
    cout << "7. Pindah posisi pointer ke berikutnya." << endl;
    cout << "8. Pindah posisi pointer ke sebelumnya." << endl;
    cout << endl << " Pilihan >> ";
    cin >> option;

  switch (option)
   {
   case 1 : tambah_awal_list();
     break;
   case 2 : menambah_node_di_akhir();
     break;
   case 3 : tambah_tengah_list();
     break;
   case 4 : hapus_awal_node();
     break;
   case 5 : hapus_akhir_node();
     break;
   case 6 : Hapus_tengah_list();
     break;
   case 7 : pindah_posisi_sebelumnya();
     break;
   case 8 : pindah_posisi_berikutnya();
   }
  }
  while (option != 0);
  }
```

8.2 DOUBLE LINKED LIST

Pada dasarnya, penggunaan Double Linked List hampir sama dengan penggunaan Single Linked List yang telah kita pelajari pada materi sebelumnya.

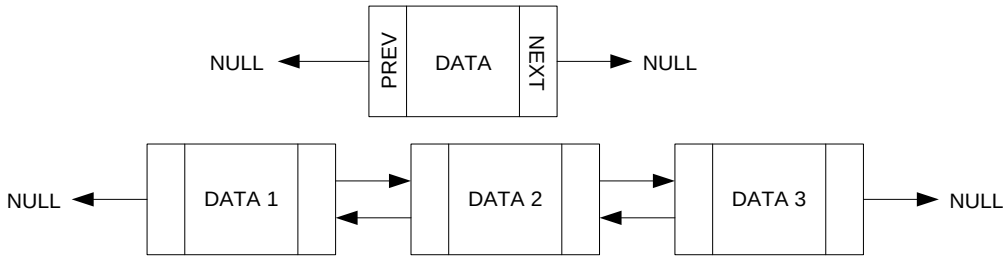
Hanya saja Double Linked List menerapkan sebuah pointer baru, yaitu **prev**, yang digunakan untuk menggeser mundur selain tetap mempertahankan pointer **next**.

a. Keberadaan 2 pointer penunjuk (**next** dan **prev**) menjadikan Double Linked List menjadi lebih fleksibel dibandingkan Single Linked List, namun dengan mengorbankan adanya memori tambahan dengan adanya pointer tambahan tersebut.
b. Ada 2 jenis Double Linked List, yaitu: Double Linked List Non Circular dan Double Linked List Circular.

A. DLLNC

- DLLNC adalah sebuah Linked List yang terdiri dari dua arah pointer, dengan node yang saling terhubung, namun kedua pointernya menunjuk ke NULL.
- Setiap node pada linked list mempunyai field yang berisi data dan pointer yang saling berhubungan dengan node yang lainnya.

a. GAMBARAN NODE DLLNC

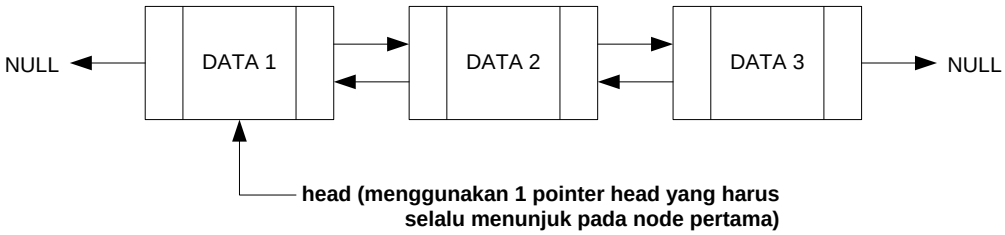


b. PEMBUATAN DLLNC

- **Deklarasi Node**

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

- **Pembuatan DLLNC dengan Head**
  - Ilustrasi :



- Fungsi-fungsi yang biasa digunakan :
  - ✓ Fungsi untuk inisialisasi awal

```
void init()// inisialisasi awal
{
        TNode *head;
        head = NULL;
}
```

  - ✓ Perlu diperhatikan :
    - Fungsi ini harus ada, untuk memunculkan node awal.
    - Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini. Jangan lupa untuk mendeklarasikan node-nya terlebih dahulu.

  - ✓ Fungsi untuk mengecek kosong tidaknya Linked List

```
int isEmpty()// mengecek kosong tidaknya Linked List
{
        if(head == NULL)
                return 1;
        else
                return 0;
}
```

  - ✓ Perlu diperhatikan :
    - Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

  - ✓ Fungsi untuk menambahkan data di depan

```
void insertDepan(int value)// penambahan data di depan
{
        TNode *baru;
        baru = new TNode;// pembentukan node baru

        baru->data = value;// pemberian nilai terhadap data baru
        baru->next = NULL;// data pertama harus menunjuk ke NULL
```

```
        baru->prev = NULL;// data pertama harus menunjuk ke NULL

        if(isEmpty() == 1)// jika Linked List kosong
        {
            head = baru;// head harus selalu berada di depan
            head->next = NULL;
            head->prev = NULL;
        }
        else // jika Linked List sudah ada datanya
        {
            baru->next = head;// node baru dihubungkan ke head
            head->prev = baru;// node head dihubungkan ke node baru
            head = baru;// head harus selalu berada di depan
        }

        printf("data masuk\n");
}
```

- ✓ Perlu diperhatikan :
- Baca code beserta panduan proses yang terjadi, pahami, lalu gambarkan ilustrasi proses terjadinya penambahan di depan. Misalkan saja data pada Linked List ada 4.
- Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

- ✓ Fungsi untuk menambahkan data di belakang

```
void insertBelakang(int value)// penambahan data di belakang
{
        TNode *baru, *bantu;
        baru = new TNode;// pembentukan node baru

        baru->data = value; // pemberian nilai terhadap data baru
        baru->next = NULL;// data pertama harus menunjuk ke NULL
        baru->prev = NULL;// data pertama harus menunjuk ke NULL

        if(isEmpty() == 1)// jika Linked List kosong
        {
            head = baru;//head harus selalu berada di depan
            head->next = NULL;
            head->prev = NULL;
        }
        else
        {
            bantu = head;// bantu diletakan di head dulu
            while(bantu->next != NULL)
```

```
            {
                bantu = bantu->next  // menggeser hingga node
                                terakhir
            }

            baru->next = baru; // node baru dihubungkan ke head
            head->prev = bantu; // node head dihubungkan
                            ke node baru
        }
        printf("data masuk\n");
}
```

- ✓ Perlu diperhatikan :
- Jika Linked List hanya menggunakan head, maka dibutuhkan satu pointer untuk membantu mengetahui node terakhir dari Linked List. Dalam code di atas digunakan pointer bantu.
- Baca code beserta panduan proses yang terjadi, pahami, lalu gambarkan ilustrasi proses terjadinya penambahan di belakang. Misalkan saja data pada Linked List ada 4.
- Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

- ✓ Fungsi untuk menambahkan data di tengah (menyisipkan data)

```
void insertTengah(int value, int cari)//penambahan data di tengah
{
        TNode *baru, *bantu, *bantu2;
        baru = new TNode;// pembentukan node baru

        baru->data = value; // pemberian nilai terhadap data baru
        baru->next = NULL;// data pertama harus menunjuk ke NULL
        baru->prev = NULL;// data pertama harus menunjuk ke NULL
        bantu = head; // bantu diletakan di head dulu

        while(bantu->data != cari)
        {
            bantu = bantu->next;//menggeser hingga didapat data cari
        }

        bantu2 = bantu->next;// menghubungkan ke node setelah
        yang dicari
        baru->next = bantu2; // menghubungkan node baru
        bantu2->prev = baru;
        bantu->next = baru;// menghubungkan ke node
```

```
                    sebelum yang dicari
        baru->prev = bantu;
}
```

- ✓ Perlu diperhatikan :
- Dibutuhkan satu pointer untuk membantu mencari node di mana data yang ingin disisipkan ditempatkan. Dalam code di atas digunakan pointer bantu.
- Penggunaan pointer bantu2 pada code di atas sebenarnya bisa digantikan dengan pemanfaatan pointer bantu. Bagaimana caranya?
- Baca code beserta panduan proses yang terjadi, pahami, lalu gambarkan ilustrasi proses terjadinya penambahan di tengah. Misalkan saja data pada Linked List ada 4, lalu sisipkan data baru setelah node kedua.
- Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

- ✓ Fungsi untuk menghapus data di depan

```
void deleteDepan()// penghapusan data di depan
{
        TNode *hapus;

        if(isEmpty() == 0)// jika data belum kosong
        {
            if(head->next != NULL)// jika data masih lebih dari 1
            {
                hapus = head;// letakan hapus pada head
                head = head->next;// menggeser head
                head->prev = NULL; // head harus menuju ke NULL
                delete hapus;//proses delete tidak boleh
                dilakukan jika node masih ditunjuk oleh pointer
            }
            else// jika data tinggal head
            {
                head = NULL; // langsung diberi nilai NULL saja
            }
            printf("data terhapus\n");
        }
        Else  // jika data sudah kosong
            printf("data kosong\n");
}
```

- ✓ Perlu diperhatikan :

- Dibutuhkan satu pointer untuk membantu memindahkan head ke node berikutnya. Dalam code di atas digunakan pointer hapus. Mengapa head harus dipindahkan?
- Baca code beserta panduan proses yang terjadi, pahami, lalu gambarkan ilustrasi proses terjadinya penghapusan di depan. Misalkan saja data pada Linked List ada 4.
- Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

✓ Fungsi untuk menghapus data di belakang

```
void deleteBelakang()// penghapusan data di belakang
{
        TNode *hapus;

        if(isEmpty() == 0) // jika data belum kosong
        {
            if(head->next != NULL) // jika data masih lebih dari 1
            {
                hapus = head; // letakan hapus pada head
                while(hapus->next != NULL)
                {
                    hapus = hapus->next;// menggeser hingga node akhir
                }

                hapus->prev->next = NULL;// menghubungkan node
                 sebelumnya dengan NULL
                delete hapus;//proses delete tidak boleh
                dilakukan jika node sedang ditunjuk oleh pointer
            }
            else // jika data tinggal head
            {
                head = NULL;// langsung diberi nilai NULL saja
            }
            printf("data terhapus\n");
        }
```

✓ else// jika Perlu diperhatikan :

- Jika Linked List hanya menggunakan head, maka dibutuhkan satu pointer untuk membantu mengetahui node terakhir dari Linked List. Dalam code di atas digunakan pointer hapus.
- Jangan lupa untuk tetap mengaitkan node terakhir ke NULL.

- Baca code beserta panduan proses yang terjadi, pahami, lalu gambarkan ilustrasi proses terjadinya penghapusan di belakang. Misalkan saja data pada Linked List ada 4.
- Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

✓ Fungsi untuk menghapus data di tengah

```
        data sudah kosong
            printf("data kosong\n");

}
void deleteTengah(int cari)// penghapusan data di tengah
{
        TNode *hapus, *bantu, *bantu2;

        hapus = head;// letakan hapus pada head
        while(hapus->data != cari)
        {
            hapus = hapus->next;// menggeser hingga data cari
        }
        bantu2 = hapus->next;// mengkaitkan node
                        sebelum dan sesudahnya
        bantu = hapus->prev;
        bantu->next = bantu2;
        bantu2->prev = bantu;
        printf("data terhapus\n");
        delete hapus; //proses delete tidak boleh dilakukan jika node sedang
ditunjuk oleh pointer
}
```

✓ Perlu diperhatikan :

- Dibutuhkan satu pointer untuk membantu mencari node di mana data yang ingin dihapus ditempatkan. Dalam code di atas digunakan pointer hapus.
- Penggunaan pointer bantu dan bantu2 pada code di atas sebenarnya bisa digantikan dengan pemanfaatan pointer hapus. Bagaimana caranya?
- Baca code beserta panduan proses yang terjadi, pahami, lalu gambarkan ilustrasi proses terjadinya penghapusan di tengah. Misalkan saja data pada Linked List ada 4, lalu hapus data pada node ketiga.

- Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

✓ Fungsi untuk menghapus semua data

```
void clear()// penghapusan semua data
{
        TNode *bantu, *hapus;
        bantu = head;// letakan bantu pada head
        while (bantu != NULL)// geser bantu hingga akhir
        {
              hapus = bantu;
              bantu = bantu->next;
              delete hapus;// delete satu persatu node
        }

        head = NULL;// jika sudah habis berikan nilai NULL pada head
}
```

✓ Perlu diperhatikan :

- Dibutuhkan dua pointer untuk membantu menggeser dan menghapus, di mana dalam code di atas digunakan pointer bantu dan hapus.
- Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

✓ Fungsi untuk menampilkan semua data

```
void cetak()// menampilkan semua data
{
        TNode *bantu;
        bantu = head;// letakan bantu pada head

        if(isEmpty() == 0)
        {
              while (bantu != NULL)
              {
                    printf("%d ", bantu->data);// cetak data pada setiap node
                    bantu = bantu->next;// geser bantu hingga akhir
              }
              printf("\n");
        }
        else// jika data sudah kosong
              printf("data kosong");
}
```

- ✓ Perlu diperhatikan :
- Dibutuhkan satu pointer untuk membantu menggeser, di mana dalam code di atas digunakan pointer bantu.
- Setelah memahami penggunaan fungsi ini, bukalah Turbo C++ Anda, dan copy-kan fungsi ini.

BAB IX
GRAPH

Graf adalah kumpulan noktah (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan garis (sisi). *Graph* dapat digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari *graph* adalah dengan menyatakan objek sebagai noktah, bulatan atau titik (Vertex), sedangkan hubungan antara objek dinyatakan dengan garis (Edge).

G = (V, E)

Dimana

G = Graph

V = Simpul atau Vertex, atau Node, atau Titik

E = Busur atau Edge, atau arc

Graf merupakan suatu cabang ilmu yang memiliki banyak terapan. Banyak sekali struktur yang bisa direpresentasikan dengan graf, dan banyak masalah yang bisa diselesaikan dengan bantuan graf. Seringkali graf digunakan untuk merepresentasikan suaru jaringan. Misalkan jaringan jalan raya dimodelkan graf dengan kota sebagai simpul (*vertex/node*) dan jalan yang menghubungkan setiap kotanya sebagai sisi (*edge*) yang bobotnya (*weight*) adalah panjang dari jalan tersebut.[20]

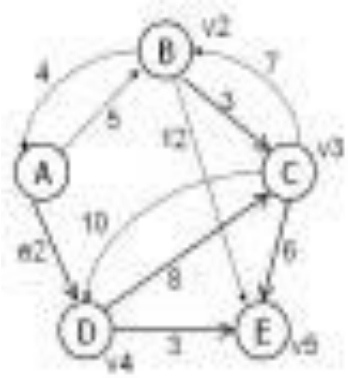
Ada beberapa cara untuk menyimpan graph di dalam sitem komputer. Struktur data bergantung pada struktur graph dan algoritma yang digunakan untuk memmanipulasi graph. Secara teori salah satu dari keduanya dapat dibedakan antara struktur list dan matriks, tetapi dalam penggunaannya struktur terbaik yang sering digunakan adalah kombinasi keduanya.

1. Graph tak berarah (undirected graph atau non-directed graph) :
   • Urutan simpul dalam sebuah busur tidak dipentingkan. Misal busur e1 dapat disebut busur AB atau BA
2. Graph berarah (directed graph) :

[20] Hariyanto, Bambang,2000,Struktur Data,Bandung

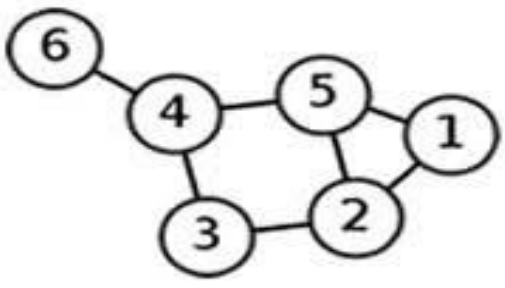
• Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.

3. Graph Berbobot (Weighted Graph)
    - Jika setiap busur mempunyai nilai yang menyatakan hubungan antara 2 buah simpul, maka busur tersebut dinyatakan memiliki bobot.
    - Bobot sebuah busur dapat menyatakan panjang sebuah jalan dari 2 buah titik, jumlah rata-rata kendaraan perhari yang melalui sebuah jalan, dll.



A. TREE

Tree dalam pemrograman merupakan struktur data yang tidak linear / non linear yang digunakan terutama untuk merepresentasikan hubungan data yang bersifat hierarkis antara elemen-elemennya. Kumpulan elemen yang salah satu elemennya disebut dengan root (akar) dan sisa elemen yang lain disebut sebagai simpul (node/vertex) yang terpecah menjadi sejumlah himpunan yang tidak saling berhubungan satu sama lain, yang disebut subtree / cabang.[21]



21

Adapun Perbedaan Graph dengan Tree sebagai berikut:

a. Pada Tree tidak terdapat Cycle

b. Pada Graph tidak memiliki root

B. ISTILAH-ISTILAH DALAM GRAF

Kemudian terdapat istilah-istilah yang berkaitan dengan graph yaitu:

a. Vertex

Adalah himpunan node / titik pada sebuah graph.

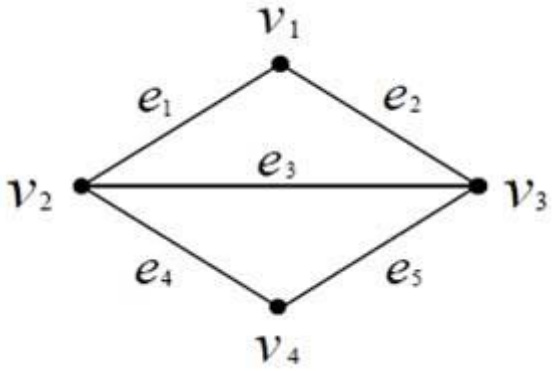
b. Edge

Adalah himpunan garis yang menghubungkan tiap node / vertex.

c. Adjacent

Adalah dua buah titik dikatakan berdekatan (*adjacent*) jika dua buah titik tersebut terhubung dengan sebuah sisi. Adalah Sisi e3 = v2v3 *insident* dengan titik v2 dan titik v3, tetapi sisi e3 = v2v3 tidak *insident* dengan titik *v1* dan titik *v4*.

Titik *v1 adjacent* dengan titik *v2* dan titik *v3*, tetapi titik *v1* tidak *adjacent* dengan titik *v4*.



d. Weight

Adalah Sebuah graf G = (V, E) disebut sebuah **graf berbobot** (*weight graph*), apabila terdapat sebuah fungsi bobot bernilai real W pada himpunan E,[22]

W : E □ R,

nilai W(e) disebut bobot untuk sisi e, □ e □ E. Graf berbobot tersebut dinyatakan pula sebagai G = (V, E, W).

Graf berbobot G = (V, E, W) dapat menyatakan

- □ suatu sistem perhubungan udara, di mana
  - □ V = himpunan kota-kota
  - □ E = himpunan penerbangan langsung dari satu kota ke kota lain
  - □ W = fungsi bernilai real pada E yang menyatakan jarak atau ongkos atau waktu
- □ suatu sistem jaringan komputer, di mana
  - □ V = himpunan komputer
  - □ E = himpunan jalur komunikasi langsung antar dua komputer
  - □ W = fungsi bernilai real pada E yang menyatakan jarak atau ongkos atau waktu

e. Path

Adalah **Walk dengan setiap vertex berbeda. Contoh, P = D5B4C2A** Sebuah **walk** (W) didefinisikan sebagai urutan (tdk nol) vertex & edge. Diawali origin vertex dan diakhiri terminus vertex. Dan setiap 2 edge berurutan adalah series. Contoh, W = A1B3C4B1A2.

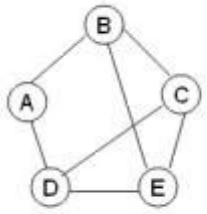
f. Cycle

Adalah Siklus (*Cycle*) atau Sirkuit (*Circuit*) Lintasan yang berawal dan berakhir pada simpul yang sama

C. REPRESENTASI GRAPH DALAM BENTUK MATRIX:

1. Adjacency Matrik Graf Tak Berarah

[22] Wirth, N,1986,Algorithms & Data Stuctures, Prentice Hall



Gambar 1a

	A 0	B 1	C 2	D 3	E 4
A 0	0	1	0	1	0
B 1	1	0	1	0	1
C 2	0	1	0	1	1
D 3	1	0	1	0	1
E 4	0	1	1	1	0

Gambar 1b

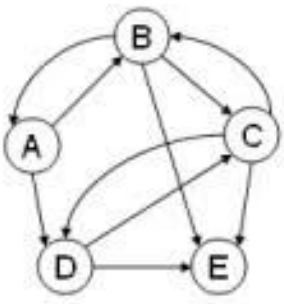
Matrik yang digambarkan pada gambar 1b merupakan representasi dalam bentuk Adjacency Matrik dari graf yang digambarkan pada gambar 1a. Beberapa hal yang dapat dilihat atau dapat diterangkan pada Adjacency Matrik tersebut adalah sebagai berikut :

1. Matrik yang terbentuk adalah matrik bujur sangkar n x n, dimana n = jumlah simpul yang ada dalam graf tersebut. Matrik ini menyatakan hubungan antara simpul satu dengan simpul lainnya.

2. Matrik yang terbentuk adalah matrik simetris dengan sumbu simetris adalah diagonal dari titik kiri atas ke titik kanan bawah.

3. Data yang tedapat baik dalam baris maupun kolom, dapat menyatakan degree sebuah simpul. Contoh : baik pada baris D maupun kolom D jumlah angka 1 nya adalah 3 buah, dimana jumlah ini menyatakan degree simpul D.[23]

2. Adjacency Matrik Graf Berarah



Gambar 2a

	A 0	B 1	C 2	D 3	E 4
A 0	0	1	0	1	0
B 1	1	0	1	0	1
C 2	0	1	0	1	1
D 3	0	0	1	0	1
E 4	0	0	0	0	0

Gambar 2b

[23] Wirth, N,1986,Algorithms & Data Stuctures, Prentice Hall

Matrik yang digambarkan pada gambar 2b merupakan representasi dalam bentuk Adjacency Matrik dari graf yang digambarkan pada gambar 2a. Beberapa hal yang dapat dilihat atau dapat diterangkan pada Adjacency Matrik tersebut adalah sebagai berikut :

1. Matrik yang terbentuk adalah matrik bujur sangkar n x n, dimana n = jumlah simpul yang ada dalam graf tersebut. Matrik ini menyatakan hubungan antara simpul satu dengan simpul lainnya.

2. Matrik yang terbentuk mungkin simetris mungkin juga tidak simetris. Menjadi Simetris bila hubungan antara dua buah simpul (v1 dan v2) terdapat busur dari v1 ke v2 dan juga sebaliknya.

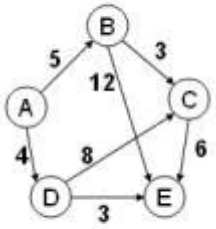
3. Hal pokok yang dinyatakan oleh matrik ini adalah : busur yang ’keluar’ dari suatu simpul. Dengan demikian, data yang terdapat dalam suatu baris, dapat menyatakan outdegree simpul yang bersangkutan.

   Contoh : Jum lah elemen yang nilainya = 1 pada baris B ada 3 elemen,ini menyatakan jumlah outdegree simpul B adalah 3 buah.

4. Data yang terdapat dalam suatu kolom, dapat menyatakan indegree simpul bersangkutan.

   Contoh : Jumlah elemen yang nilainya 1 pada kolom B ada 2 elemen, ini menyatakan indegree simpul B adalah 2 buah.

3. Adjacency Matrik Graf Berbobot Tak Berarah



Gambar 3a

	A 0	B 1	C 2	D 3	E 4
A 0	0	5	999	4	999
B 1	5	0	3	999	12
C 2	999	3	0	8	6
D 3	4	999	8	0	3
E 4	999	12	6	3	0

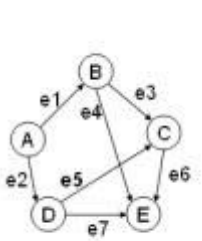
Gambar 3b

Nilai yang ada dalam tiap elemen matrik, menyatakan bobot busur yang menghubungkan dua buah simpul yang bersangkutan. Untuk dua buah simpul yang tidak berhubungan langsung oleh sebuah busur, maka dianggap dihubungkan oleh sebuah busur yang nilai bobotnya tidak terhingga. Dalam pemograman, karena keperluan algoritma, maka dari total bobot seluruh busur yang ada atau yang mungkin ada.

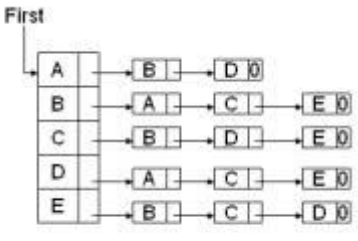
Contoh: pada gambar 3a simpul A dan C tidak berhubungan langsung melalui sebuah busur, maka untuk elemen matrik yang bersangkutan diisi dengan nilai 999 karena nilai 999 dalam kasus ini cukup mewakili nilai tidak terhingga.

D. REPRESENTASI GRAF DALAM BENTUK LINKED LIST

1. Adjacency List



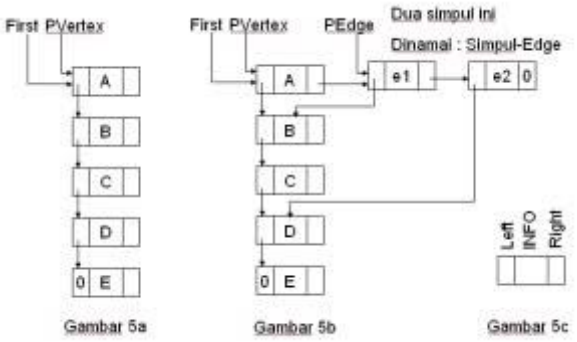
Gambar 4a



Gambar 4b

Bila ingin direpresentasikan dalambentuk linked list, dapat diilustrasikan secara sederhana seperti gamabar 4b. Dari ilustrasi sederhana tersebut

terlihat ada 5 buah simpul A,B,C,D,dan E yang dibariskan dari atas kebawah seperti pada gambar 4a. Kemudian dari masing-masing simpul ’keluar’ pointer kearah kanan yang menunjuk simpul-simpul lain. Salah satu contoh, yang dapat dilihat pada gambar 4b dimana A menunjuk simpul B dan simpul D.



Gambar 5a Gambar 5b Gambar 5c

Dalam Adjacency List, kita perlu membedakan antara simpul-vertex dan simpul-edge. Simpul-vertex untuk menyatakan simpul atau vertex, dan simpul-edge untuk menyatakan hubungan antar simpul yang biasa disebut busur. Struktur keduanya bisa sama, bisa juga tidak sama,tergantung kebutuhan.Untuk memudahkan pembuatan program, struktur kedua macam simpul dibuat sama seperti yang digambarkan pada gambar 5c. Yang membedakan antara simpul-vertex dan simpul-edge, adalah anggapan terhadap simpul tersebut. Dalam contoh ini, terlihat struktur simpul dibuat terdiri dari 3 elemen. Satu elemen untuk INFO, dua elemen untuk pointer.pointer kiri (left) dan pointer kanan (right).

```
Struct tipes{

Struct tipes *Left;

int INFO;

Struct tipes *Right;
```

}; Struct tipes *First,*Pvertex,*Pedge;

- Bila simpul dianggap sebagai simpul-vertex, maka :

  Pointer left digunakan untuk menunjuk simpul berikutnya dalam untaian simpul-simpul yang ada,atau diisi NULL bila sudah tidak ada simpul yang pelu ditunjuk.Sedangkan pointer Right digunakan untuk menunjuk simpul edge yang pertama.

- Bila Simpul dianggap sebagai simpul-edge, maka :

  Pointer left digunakan untuk menunjuk simpul-vertex ‘tujuan’ yang berhubungan dengan simpul-vertex ‘asal’ dan pointer right digunakan untuk menunjuk simpul-edge berkiutnya bila masih ada, atau diisi NULL bila tak ada lagi simpul-busur yang ditunjuk.

E. CONTOH REPRESENTASI GRAPH DALAM BAHASA C

```
typedef struct vertex
{ char nama[100];
float x, y;
float status;
float jarak;
struct vertex *next,*connector;
};
typedef struct vertex *pvertex;
typedef struct edge
{
```

```
float jarak;
char titik1[100];
char titik2[100];
edge * next;
};
typedef struct edge * garis;
garis awalga = NULL, akhirga= NULL;
pvertex makeptree (float x, float y, char nama[])
{
pvertex baru;
baru=(pvertex)malloc(sizeof(struct vertex));
baru->x=x;
baru->y=y;
baru->status=0;
baru->next=NULL;
baru->connector=NULL;
strcpy(baru->nama,nama);
}
void makevertex (pvertex *vertex,float x,float y,char nama[])
```

```
{
pvertex p , q ;
p = makeptree(x,y,nama);
q = *vertex;
if(q == NULL)
*vertex = p ;
else
{
for(;q->next;q=q->next)
{
}
q->next = p;
} }
void makeedge(garis * awalga, garis * akhirga, char titik1[], char titik2[], float jarak)
{
garis baru;
baru=(garis)malloc(sizeof(edge));
baru->jarak=jarak;
strcpy(baru->titik1,titik1);
```

```
strcpy(baru->titik2, titik2);
if(*awalga==NULL)
{
*awalga=baru;
*akhirga=baru;
}
else
{
(*akhirga)->next=baru;
(*akhirga)=baru;
}
}
void cek(pvertex vertex, int jumlah)
{
pvertex awal,p,q;
float jarak;
float min ;
float temp;
awal = vertex;
```

```
p = awal;
for(int i=0;i
{
min = 999;
p->status =1;
for(q=awal->next;q!=NULL;q=q->next)
{
if(q->status!=1)
{
jarak=(((p->x)-(q->x))*((p->x)-(q->x)))+(((p->y)-(q->y))*((p->y)-(q->y)));
jarak=sqrt(jarak);
if (min>jarak)
{
min = jarak;
p->jarak = min;
p->connector = q;
}
makeedge(&awalga,&akhirga,p->nama,q->nama,p->jarak);
}
```

```
}

p = p->connector;

}

temp=(((p->x)-(awal->x))*((p->x)-(awal->x)))+(((p->y)-(awal->y))*((p->y)-(awal->y)));

p->jarak = sqrt(temp);

p->connector = awal;

}

void displayedge(pvertex vertex,int jumlah)

{

pvertex list = vertex;

float tot;

tot += vertex->jarak;

printf("\n\t%s ke %s\n", list->nama,list->connector->nama);

list = list->connector;

for(int a=0; a

{

printf("\n\t%s ke %s\n", list->nama,list->connector->nama);

tot += list->jarak;

list = list->connector;
```

```
}

printf("\n");

}
```

F. KAITAN SHORHEST PATH PROBLEM DALAM GRAF

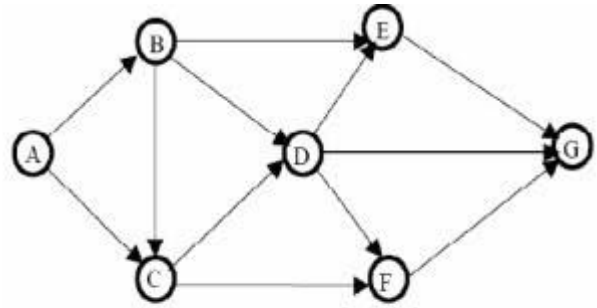
Lintasan terpendek merupakan salah satu dari masalah yang dapat diselesaikan dengan graf. Jika diberikan sebuah graf berbobot, masalah lintasan terpendek adalah bagaimana kita mencari sebuah jalur pada graf yang meminimalkan jumlah bobot sisi pembentuk jalur tersebut.

Terdapat beberapa macam persoalan lintasan terpendek antara lain:

- Lintasan terpendek antara dua buah simpul tertentu (*a pair shortets path*).
- Lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).
- Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shoertest path*).
- Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

Jalur terpendek adalah suatu jaringan pengarahan perjalanan dimana sesorang pengarah jalan ingin menentukan jalur terpendek antara dua kota, ber[24]dasarkan beberapa jalur alternatif yang tersedia, dimana titik tujuan hanya satu. Gambar 2.6 menunjukkan suatu graf ABCDEFG yang berarah dan tidak berbobot.

[24] Aho, Hopcroft, Ullman,1987,Data Structures and Algorithms,Prentice Hall



Gambar 2.6 Graf ABCDEFG

Pathing Algorhitm

Pathing Algorithm adalah sebuah algoritma yang digunakan untuk mencari suatu solusi dalam menentukan lintasan terpendek dari suatu graf. Saat ini terdapat banyak sekali algoritma yang digunakan untuk menyelesaikan persoalan lintasan terpendek diantaranya Algoritma *Djikstra* (djikstra algorithm) dan Algoritma *Bellman-Ford* (bellman-ford algorithm).

Algoritma Djikstra

Algoritma Dijkstra, dinamai menurut penemunya, Edsger Dijkstra, adalah algoritma dengan prinsip greedy yang memecahkan masalah lintasan terpendek untuk sebuah graf berarah dengan bobot sisi yang tidak negatif.

Algoritma Dijkstra merupakan salah satu varian bentuk algoritma populer dalam pemecahan persoalan yang terkait dengan masalah optimasi. Sifatnya sederhana dan lempang (straightforward). Sesuai dengan arti greedy yang secara harafiah berarti tamak atau rakus - namun tidak dalam konteks negatif -, algoritma greedy ini hanya memikirkan solusi terbaik yang akan diambil pada setiap langkah tanpa memikirkan konsekuensi ke depan.

Input algoritma ini adalah sebuah graf berarah yang berbobot (*weighted directed graph*) *G* dan sebuah sumber *vertex* *s* dalam *G* dan *V* adalah himpunan semua vertices dalam graph *G*.

Setiap sisi dari graf ini adalah pasangan vertices (*u*,*v*) yang melambangkan hubungan dari *vertex u* ke *vertex v*. Himpunan semua tepi disebut *E*. Bobot (*weights*) dari semua sisi dihitung dengan fungsi : $w: E \rightarrow [0, \infty)$, jadi $w(u,v)$ adalah jarak tak-negatif dari vertex *u* ke vertex *v*. Ongkos (*cost*) dari sebuah

sisi dapat dianggap sebagai jarak antara dua *vertex*, yaitu jumlah jarak semua sisi dalam jalur tersebut. Untuk sepasang vertex *s* dan *t* dalam *V*, algoritma ini menghitung jarak terpendek dari *s* ke *t*.

Algoritma Bellman-Ford

Algoritma Bellman-Ford menghitung jarak terpendek (dari satu sumber) pada sebuah digraf berbobot. Maksudnya dari satu sumber ialah bahwa ia menghitung semua jarak terpendek yang berawal dari satu titik node. Algoritma Dijkstra dapat lebih cepat mencari hal yang sama dengan syarat tidak ada sisi (edge) yang berbobot negatif. Maka Algoritma Bellman-Ford hanya digunakan jika ada sisi berbobot negatif.

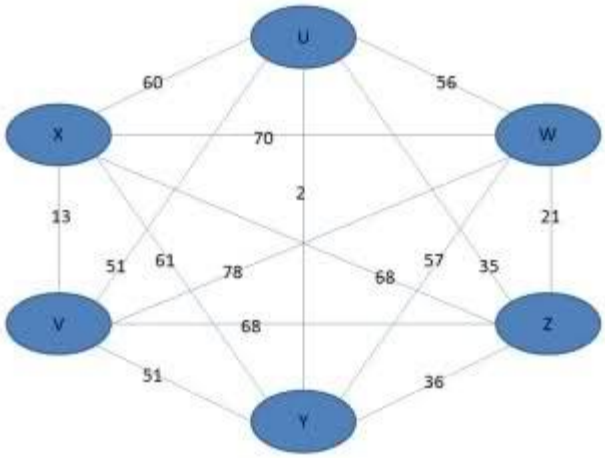
Algoritma Bellman-Ford menggunakan waktu sebesar O(V.E), di mana V dan E adalah menyatakan banyaknya sisi dan titik. Dalam konteks ini, bobot ekivalen dengan jarak dalam sebuah sisi.

Algoritma tentang TSP,CPP,Coloring Graph dan MST

a. Travelling Salesman Problem

Traveling Salesman Problem (TSP) pertama kali diperkenalkan oleh Rand pada tahun 1948, reputasi Rand membuat TSP dikenal dengan baik dan menjadi masalah yang populer. TSP merupakan persoalan yang mempunyai konsep sederhana dan mudah dipahami. Permasalahan TSP *(Traveling Salesman Problem*) adalah permasalahan dimana seorang *salesman* harus mengunjungi semua kota dimana tiap kota hanya dikunjungi sekali, dan dia harus mulai dari dan kembali ke kota asal. Pada TSP, optimasi yang diinginkan agar ditemukan rute perjalanan terpendek untuk melewati sejumlah kota dengan jalur tertentu sehingga setiap kota hanya terlewati satu kali dan perjalanan diakhiri dengan kembali ke kota semula. Pendekatan dengan menggunakan Jaringan Saraf Kohonen Self Organizing memberikan solusi atau penyelesaian dalam perhitungan waktu yang lebih singkat dibandingkan dengan sejumlah algoritma lain yang diterapkan pada

komputer dalam bentuk program. ***The Travelling Salesman Problem*** adalah sebuah bentuk permasalahan riil yang mempunyai bentuk permodelan weight hamiltonian cycle. Dalam permodelan ini, seorang salesman dalam pekerjaannya diharuskan untuk mengunjungi pelanggannya yang berada pada beberapa kota yang tiap-tiap kotaknya mempunyai jarak berbeda-beda. Untuk mengefisiensikan beban kerjanya, salesman tersebut, dia harus merancang sebuah rute perjalanan dengan waktu minimal. Selain itu, demi efisiensi biaya dan waktu, salesman tersebut berusaha agar dia tidak melewati satu kota yang sama dua kali. Berikut ini adalah gambar permodelan dari problem travelling salesman problem.



Berikut adalah Algoritma untuk mencari TSP:

□□□□□□ Mula-mula dimulai dari U cari nilai Edge yang terkecil dan masih belum dilewati dalam rute. Maka dipilih Edge U-Y yang mempunyai nilai 2. Kemudian tandai titik U dan Y agar tidak dilewati lagi.

□□□□□□ Kemudian dari titik Y pilih nilai edge terkecil dimana vertex tujuannya belum ditandai/tidak menimbulkan cycle. Dalam hal ini dipilih edge Y-Z, kemudian tandai vertex Z.

· Sama dengan step 1 dan step 2. Dalam hal ini, dipilih edge Z-W sebagai rute.

- Pada vertex W, Nilai edge yang terkecil ada pada edge U-W, tetapi karena edge ini menimbulkan cycle, maka edge ini tidak dapat digunakan, demikian halnya dengan edge W-Y, sehingga yang dipilih sebagai rute adalah Edge W-X
- Karena semua vertex sudah terhubung dan tidak ada cycle, maka path pada gambar 5 adalah solusi dari problem untuk titik U

- Jadi, yang akan dilalui adalah Path : U – Y – Z – W – X – V

DAFTAR PUSTAKA

Aho, Hopcroft, Ullman, 1987,*Data Structures and Algorithms*,Prentice Hall

Horowitz, E. & Sahni, S,1984, *Fundamentals of Data Structures in Pascal*, Pitman Publishing Limited.

Hariyanto, Bambang, 2000,*Struktur Data*,Bandung

Sjukani, Moh,2012,*Struktur Data (Algoritma dan Struktur Data dengan C,C++*,Jakarta:Mitra Wacana Media

Wirth, N.*Algorithms & Data Stuctures,1986*. Prentice Hall,1986