

**MODUL PRAKTIKUM STRUKTUR DATA  
BAHASA PEMROGRAMAN C++  
POLITEKNIK TEKNOKRAT INTERNASIONAL KOLAKA**



**DOSEN PENGAMPUH  
MARDIANTO, S.Kom., M.Cs**

## DAFTAR ISI

|  |          |
|--|----------|
| DAFTAR ISI .....                                     | ii       |
| BAB 1 STRUKTUR DATA DENGAN C++ .....                 | 1        |
| A. Pendahuluan .....                                 | 1        |
| B. Tipe Data .....                                   | 1        |
| C. Variabel .....                                    | 2        |
| D. Konstanta.....                                    | 3        |
| E. OPERATOR.....                                     | 4        |
| 1. Operator Aritmatika (+, -, *, /, %) .....         | 4        |
| 2. Operator Increment dan Decrement (++ dan --)..... | 4        |
| 3. Operator Relasional .....                         | 5        |
| 4. Operator Logika.....                              | 5        |
| 5. <b>Operator Kombinasi.....</b>                    | <b>6</b> |
| BAB 2 ARRAY .....                                    | 8        |
| A. Array 1 (satu) Dimensi .....                      | 8        |
| B. Array 2 (dua) Dimensi.....                        | 11       |
| 1. Array 2 dimensi Angka matriks.....                | 13       |
| 2. Array 2 dimensi dengan Penjumlahan.....           | 14       |
| BAB 3 STRUCTUR.....                                  | 16       |
| BAB 4 SORT .....                                     | 21       |
| A. Buble sort (Gelembung) .....                      | 21       |
| 3. Konsep algoritma bubble sort.....                 | 21       |
| B. Selection sort (Maksimum/Minimum).....            | 24       |
| 1. Konsep algoritma selection sort .....             | 24       |
| 2. Implementasi algoritma selection sort.....        | 25       |
| 3. Procedure bubble sort .....                       | 25       |
| C. Insertion sort (sisip).....                       | 26       |
| 1. Konsep algoritma insertion sort.....              | 26       |
| 2. Implementasi Insertion sort .....                 | 26       |
| 3. Procedure Insertion sort.....                     | 27       |
| BAB 5 LINKED LIST .....                              | 28       |
| B. Procedure linked list.....                        | 33       |

|                                 |    |
|---------------------------------|----|
| BAB 6 STACK.....                | 37 |
| A. Pendeklarasian stack .....   | 38 |
| B. Procedure code stack.....    | 38 |
| C. Penjelasan Code Program..... | 41 |
| BAB 7 QUEUE .....               | 45 |
| BAB 8 TREE .....                | 50 |
| BAB 9 GRAF .....                | 56 |

# BAB 1

## STUKUTUR DATA DENGAN C++

### A. Pendahuluan

Definisi data adalah kenyataan atau fakta yang tercatat mengenai suatu obyek. Pengertian data ini menyiratkan suatu nilai yang bisa dinyatakan dalam bentuk konstanta atau variabel. Struktur data adalah cara menyimpan atau merepresentasikan data didalam komputer agar bisa dipakai secara efisien dengan memanfaatkan memori/RAM komputer atau koleksi dari suatu variabel. Struktur data biasa dipakai untuk mengelompokkan beberapa informasi yang berkaitan menjadi sebuah kesatuan. Struktur data diperlukan dalam rangka membuat program komputer. Untuk menyusun sebuah program komputer diperlukan tiga macam komponen dasar yaitu Algoritma, Bahasa Pemrograman dan Struktur data.

Aspek yang berkaitan dengan algoritma biasa dikenal dengan ukuran algoritma atau efisiensi algoritma. Ukuran algoritma ditentukan oleh dua hal yaitu:

1. Efisiensi waktu
2. Efisiensi memori.

Aspek yang berkaitan dengan bahasa pemrograman terdiri dari:

1. Sintaks
2. Reserved word
3. Function
4. Procedure

Aspek yang berkaitan dengan Struktur data terdiri dari:

1. Nilai data (data value) numerik dan non numerik
2. Relasi Antar data
3. Procedure/ fungsi atau operasi pada data

Sekilas tentang C++ merupakan kembangan dari bahasa pemrograman C. sekitar tahun 1972 *Brian W. Kerninghan Dennins M Richie* menciptakan aplikasi C, sekitar beberapa dekade setelahnya *Bjarne Stroustrup* dari laboratorium bell, AT&T mengembangkan bahasa pemrograman C++, cukup kompatibel dengan bahasa pendahulunya C. Bahasa C dan C++ merupakan bahasa yang sangat populer dari pengembang perangkat lunak. Kedua bahasa digolongkan kedalam bahasa tingkat menengah (*middle level language*). Kode bahasa pemrogramannya bersifat *case sensitive*, artinya membedakan antara huruf capital dan huruf kecil. Keistimewaan dari bahasa C++ bahasa sudah didukung OOP (*Object Oriented Programming*).

### B. Tipe Data

Tipe data merupakan *identifier* atau pengenal suatu variabel. Tipe data akan memberitahukan kepada compiler mengenai jenis tipe data dan seberapa lebar compiler mengalokasikan ruang memori untuk suatu variabel. Sehingga dengan mekanisme alokasi memori pada pemrograman C++, program yang berjalan akan lebih efisien dari segi penggunaan memori.

Tipe data dalam pemrograman terkhusus C++ dapat dibedakan berdasarkan adanya tanda (signed) dan tidak adanya tanda (unsigned). Perbedaan antara keduanya adalah adanya tanda bilangan yang menunjukkan positif dan negatif. Untuk tipe data *unsigned* (tanpa tanda) suatu bilangan hanya diawali dari 0 ke suatu jangkauan tertentu, sedangkan untuk tipe *signed* (bertanda), bilangan diawali dari nilai negatif (-) menuju ke jangkauan nilai positif (+).

Tabel 1.1 Tabel Tipe data (*signed*)

| <b>Tipe Data</b> | <b>Deskripsi</b>   | <b>Ukuran Memori</b> | <b>Jangkauan</b>                                       |
|------------------|--|----------------------|--|
| int              | Bilangan bulat   | 2 byte               | -32768 hingga 32768                                    |
| short int        | Sama dengan int namun jangkauannya lebih pendek                | 2 byte               | -32768 hingga 32768                                    |
| long int         | Memiliki jangkauan lebih panjang dari int                      | 4 byte               | -2147483648 hingga 2147483648                          |
| bool             | Tipe data untuk menampung nilai kebenaran (flag)               | 1 byte               | 1 atau 0 (True atau False)                             |
| float            | Bilangan floating point atau koma                              | 4 byte               | $3,4 \times 10^{-38}$ hingga $3,4 \times 10^{+38}$     |
| double           | Sama dengan float namun memiliki jangkauan dua kali dari float | 8 byte               | $1,7 \times 10^{-308}$ hingga $1,7 \times 10^{+308}$   |
| long double      | Sama dengan double namun memiliki jangkauan lebih lebar        | 10 byte              | $3,4 \times 10^{-4932}$ hingga $3,4 \times 10^{+4932}$ |
| char             | Menampung tipe karakter  | 1 byte               | -128 hingga 128  |
| wchar_t          | Sama dengan char namun memiliki jangkauan lebih lebar          | 2 byte hingga 4 byte | 1 wide character                                       |

Tabel 1.2 Tabel Tipe data (*unsigned*)

| <b>Tipe Data</b>   | <b>Deskripsi</b>                         | <b>Ukuran Memori</b> | <b>Jangkauan</b> |
|--------------------|--|----------------------|------------------|
| Unsigned char      | Menampung karakter                       | 1 byte               | 0-255            |
| Unsigned int       | Bilangan bulat tak bertanda atau positif | 4 byte               | 0-4294967295     |
| Unsigned long int  | Long integer                             | 4 byte               | 0-4294967295     |
| Unsigned short int | Short integer                            | 2 byte               | 0-65535          |

## C. Variabel

Variabel merupakan sesuatu yang akan menjadi objek penelitian. Dalam bahasa pemrograman variabel biasa digunakan untuk memberikan nama objek yang selalu diikuti oleh *tipe data*. Variabel dibedakan menjadi 2 (dua) yaitu variabel *global* dan variabel *local*. Variabel *global* dapat dipanggil dimanapun dari suatu program, sedangkan variabel *local* hanya bisa dipanggil pada

fungsi tertentu saja dan variabel tersebut tidak berdampak diluar fungsi. Untuk lebih jelasnya dapat dilihat pada baris code berikut:

```
1 #include <stdio.h>
2 int global=9; //merupakan variabel global
3
4 int main(){
5     int local=5; //merupakan variabel Local|
```

Pada *code* baris 2 diatas variabel *global* diberikan nilai 9 sedangkan variabel *local* diberi nilai 5. Sehingga kita di jalan akan tampil seperti dibawah ini.

```
□ Select E:\DATAKU\Dosan\Matakuliah\Struktur DAta\2017 2018\CODE\variabel.exe
nilai Global = 9
nilai Local  = 5
-----
Process exited after 0.1334 seconds with return value 0
Press any key to continue . . .
```

## D. Konstanta

Konstanta merupakan variabel dengan nilai tetap dan tidak dapat diubah. Pendeklarasian konstanta dapat dilakukan dengan 2 cara yaitu dengan menggunakan *const* dan *#define*.

```
1 #include <stdio.h>
2
3 const char nama[]="Almin";
4 #define alamat "Taho"
5 int main(){
6     printf("namanya = %s \n", nama);
7     printf("Alamat  = %s", alamat);
8     return 0;
9 }
```

Pada *code* baris 3 (tiga) variabel *nama* dideklarasikan dengan menggunakan konstanta *const* sedangkan *code* baris 4 (empat) *alamat* dideklarasikan dengan menggunakan konstanta *#define*. Hasilnya dapat dilihat seperti berikut:

```
□ E:\DATAKU\Dosan\Matakuliah\Struktur DAta\2017 2018\CODE\kontranta.exe
namanya = Almin
Alamat  = Taho
-----
Process exited after 0.1203 seconds with return value 0
Press any key to continue . . . ■
```

## E. OPERATOR

Operator adalah suatu symbol atau karakter yang digunakan untuk melakukan suatu operasi atau manipulasi. Dalam bahasa pemrograman C/C++ terdapat berbagai macam operator yang dapat dimanfaatkan

### 1. Operator Aritmatika (+, -, \*, /, %)

Tabel 1.3 Tabel Operator aritmatika

| Operator | Fungsi                   |
|----------|--------------------------|
| *        | Kali                     |
| /        | Bagi                     |
| %        | Sisa pembagian (Modulus) |
| +        | Tambah                   |
| -        | Kurang                   |

Untuk operator % atau modulus yaitu untuk mengetahui sisa hasil bagi. Misalnya  $a = 9 \% 2$ , maka variable a akan terisi nilai 1 karena sisa hasil bagi 9 dan 2 adalah 1.

```
#include <stdio.h>
int main(){
    int a,c,d,e;
    float b;
    a = 3*5;
    b= 5.0/3;
    c= 5+6;
    d = 6-3;
    e = 9%2;
    printf("Hasil perkalian    =%d \n", a);
    printf("Hasil Pembagian   =%.2f \n", b);
    printf("Hasil Penjumlahan =%d \n", c);
    printf("Hasil Pengurangan =%d \n", d);
    printf("Hasil Sisa bagi   =%d \n", e);
    return 0;
}
```

### 2. Operator Increment dan Decrement (++ dan --)

Operator Increment atau penaikan (++) akan menaikkan atau menambahkan 1 nilai variable. Sedangkan operator decrement (--) akan menurunkan atau mengurangi 1 nilai variable.

Tabel 1.4 Tabel Operator Increment dan Decrement

| Operator | Arti    |
|----------|---------|
| x++/++x  | $x=x+1$ |
| x--/--x  | $x=x-1$ |

```
#include <stdio.h>

int main(){
    int inc =4;
    int dsc =4;
    int hasilinc, hasildsc;
    hasilinc=++inc;
    hasildsc=--dsc;
    printf("NILAI AWAL      =4 \n");
    printf("Hasil Increment   =%d \n", hasilinc);
    printf("Hasil Descrement =%d \n", hasildsc);
    return 0;
}
```

### 3. Operator Relasional

Operator Relasional merupakan operator yang bernilai boolean bukan sebuah nilai numerik. Operator ini hanya mengenal benar atau salah.

Tabel 1.5 Tabel Operator Relasional

| Operator | Keterangan                         |
|----------|------------------------------------|
| ==       | Sama dengan                        |
| !=       | Tidak sama dengan                  |
| >        | Lebih dari                         |
| <        | Kurang dari                        |
| >=       | Lebih besar dari atau sama dengan  |
| <=       | Lebih kurang dari atau sama dengan |

Contoh :

(7 == 5) hasilnya adalah salah (false)

(7 > 5) hasilnya adalah benar (true)

Jika dilihat dari contoh diatas, operator relasional selalu diikuti fungsi percabangan atau biasa dikenal dengan operator logika.

### 4. Operator Logika

Operator logika juga digunakan untuk memberikan nilai atau kondisi benar (true) dan salah (false). Biasanya operator ini dipakai untuk membandingkan dua kondisi dengan menggunakan operator relasional. Contoh code operator logika adalah:

```
#include <stdio.h>

int main(){
    int nilai;
    printf("masukan Nilai : ");
    scanf("%d", &nilai);

    if(nilai >= 70){
        printf("Nilai lebih besar dari 70");
    }else{
        printf("Nilai lebih kecil dari 70");
    }
}
```

Pada penjelasan baris code diatas yaitu membandingkan variabel *nilai* yang dimasukan dengan nilai *konstanta* 70. Jika nilai lebih besar atau sama dengan 70 maka akan muncul pemberitahuan “nilai lebih besar dari 70” sedangkan jika dibawah dari 70 maka akan muncul “nilai lebih kecil dari 70”.

## 5. Operator Kombinasi

Operator kombinasi digunakan untuk memendekan operator penugasan, seperti contoh dapat dilihat pada tabel berikut:

Tabel 1.5 Tabel Operator Kombinasi

| Operator Kombinasi | Arti padanannya |
|--------------------|-----------------|
| X += 3             | X = X +3        |
| X -= 3             | X = X -3        |
| X *= 3             | X = X *3        |
| X /= 3             | X = X /3        |
| X %= 3             | X = X %3        |
| X <= 3             | X = X << 3      |
| X >= 3             | X = X >> 3      |
| X &= 3             | X = X & 3       |
| X  = 3             | X = X   3       |
| X ^= 3             | X = X ^ 3       |

Jika melihat kolom operator pada tabel operator kombinasi pada kolom pertama disepadankan pada kolom sebelahnya. Sebenarnya hasilnya sama namun penggunaan operator kombinasi lebih sederhana jika dibandingkan dengan padanannya. Contoh

x = x + 2;

y = y \* 4;

bisa disederhanakan menjadi

```
x +=2;  
y*=4;
```

source code lengkap dapat dilihat pada gambar berikut:

```
#include <stdio.h>  
  
int main(){  
    int a =4;  
    int b =4;  
  
    a+=2;  
    b*=4;  
    printf("NILAI AWAL      =4 \n\n");  
    printf("Hasil Operator Kombinasi Tambah =%d \n", a);  
    printf("Hasil Operator Kombinasi Kali  =%d \n", b);  
    return 0;  
}
```

## BAB 2

# ARRAY

### A. Array 1 (satu) Dimensi

Array atau larik suatu data terstruktur yang memiliki tipe data yang sama dalam urutan tertentu. Nilai-nilai data pada suatu array disebut dengan elemen-elemen larik. Letak urutan dari suatu larik ditunjukkan oleh suatu subscript atau index. Elemen-elemen dari array tersusun secara sequential dalam memori komputer. Array atau larik dapat berupa satu dimensi, dua dimensi, tiga dimensi ataupun banyak dimensi.

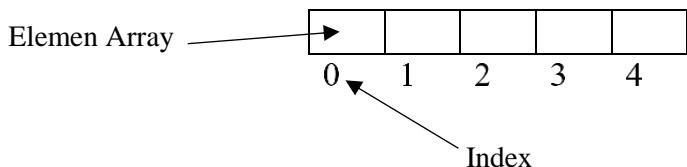
Deklarasi Array

Tipe\_data nama\_var\_array [ukuran];

int angka 5

Keterangan :

- tipe\_data : menyatakan jenis tipe data elemen larik (int, char, float, dll)  
nama\_var\_array : menyatakan nama variabel yang dipakai.  
ukuran : menunjukkan jumlah maksimal elemen larik.



Pengisian elemen array dapat dilakukan dengan tiga cara yaitu:

Pengisian variabel (inisialisasi)

Array satu dimensi dapat dilakukan dengan memberikan nilai awal saat deklarasi variabel. Bentuk umum pengisian variabel atau inisialisasi satu dimensi dapat dilihat sebagai berikut:

*Tipe\_data nama\_var\_array [ukuran] = {elemen-0, elemen-1,...,elemen-4};*

Perhitungan element dimulai dari *index* 0, dimana data dapat diisi sejumlah elemen yang disediakan. Jika elemen yang dimasukan kurang daripada ukuran maka akan diberi nilai 0 (nol) namun jika jumlah elemen lebih daripada elemen yang disediakan maka akan muncul *error "Too many initializers"*. Contoh penulisan array melalui pengisian variabel adalah seperti berikut:

int angka[5] = { 10,50,30,20,12};

Dapat dilihat pada contoh diatas bahwa kita telah memesan tempat pada memori komputer sebanyak 5 (lima) dengan index mulai dari 0 sampai 4. Dimana elemen-elemen akan dimasukan

kedalam lokasi nilai secara berturut turut. Code lengkap penggunaan array pengisian variabel (inisialisasi) dapat dilihat seperti berikut:

```
1 #include <stdio.h>
2 int main(){
3     int angka[5]={10,50,30,20,12};
4     printf("nilai Index 0 = %d \n", angka[0]);
5     printf("nilai Index 1 = %d \n", angka[1]);
6     printf("nilai Index 2 = %d \n", angka[2]);
7     printf("nilai Index 3 = %d \n", angka[3]);
8     printf("nilai Index 4 = %d \n", angka[4]);
9     return 0;
10}
11
```

Atau bisa diganti dengan menggunakan perulangan

```
1 #include <stdio.h>
2
3 int main(){
4     int angka[5]={10,50,30,20,12};
5     int i;
6
7     for (i=0; i <5; i++) {
8         printf("nilai Index %d = %d \n",i, angka[i]);
9     }
10    return 0;
11}
```

Maka hasilnya

```
nilai Index 0 = 10
nilai Index 1 = 50
nilai Index 2 = 30
nilai Index 3 = 20
nilai Index 4 = 12
```

#### ➤ Pengisian Penugasan

Elemen array satu dimensi dengan menggunakan operator penugasan dapat dilakukan dengan kode seperti dibawah ini:

nama\_var\_array [indeks/posisi] = elemen;

```
1 #include <stdio.h>
2 int main(){
3     int angka[5], i;
4     angka[0] =10;
5     angka[1] =50;
6     angka[2] =30;
7     angka[3] =20;
8     angka[4] =12;
9     for (i=0; i <5; i++) {
10         printf("nilai Index %d = %d \n",i, angka[i]);
11     }
12     return 0;
13 }
```

- Dibaca melalui media masukan

Langkah terakhir adalah dengan menggunakan media masukan, yaitu membaca data melalui media inputan (*keyboard*) langsung dari program. Cara kerja model ini dengan menampung sejumlah data inputan sesuai dengan jumlah elemen yang sudah ditentukan, kemudian dijadikan sebagai keluaran (output). Cara seperti ini sangat cocok jika menggunakan fungsi perulangan (*for, while, do while*). Berikut contoh code inputan melalui media masukan:

```
// media inputan
for(i;i<5;i++){
    printf("masukan nilai elemen index %d = ",i);
    scanf("%d",&angka[i]);
}
```

Pada baris code diatas adalah menggunakan media inputan, dimana semua nilai elemen yang dimasukan akan dikumpulkan sebanyak elemen array (contoh 5 elemen). Kemudian akan dikeluarkan (output) setelah semua elemen array sudah terinput. adapun untuk menampilkan nilai elemen:

```
11 //Laporan
12     printf("\n\nlaporan hasil nilai elemen : \n");
13     for(j;j<5;j++){
14         printf("Hasil nilai elemen index %d = %d \n",j,angka[j]);
15 }
```

Penggunaan code untuk menampilkan hasil sama dengan cara kedua yaitu menggunakan perulangan. Code lengkap pembacaan melalui media inputan adalah sebagai berikut:

```
1 #include <stdio.h>
2 int angka[5];
3 int i,j;
4 int main()
5 {
6     // media inputan
7     for(i;i<5;i++){
8         printf("masukan nilai elemen index %d = ",i);
9         scanf("%d",&angka[i]);
10    }
11 //Laporan
12     printf("\n\nlaporan hasil nilai elemen : \n");
13     for(j;j<5;j++){
14         printf("Hasil nilai elemen index %d = %d \n",j,angka[j]);
15    }
16 }
17 }
```

Maka hasilnya sebagai berikut:

```

E:\DATAKU\Dosan\Matakuliah\Struktur DAta\2017 2018\CODE\variabel_for.exe
masukan nilai elemen index 0 = 10
masukan nilai elemen index 1 = 50
masukan nilai elemen index 2 = 30
masukan nilai elemen index 3 = 20
masukan nilai elemen index 4 = 12

laporan hasil nilai elemen :
Hasil nilai elemen index 0 = 10
Hasil nilai elemen index 1 = 50
Hasil nilai elemen index 2 = 30
Hasil nilai elemen index 3 = 20
Hasil nilai elemen index 4 = 12

```

## B. Array 2 (dua) Dimensi

Array dua dimensi adalah array yang mempunyai dua subskrip yaitu baris dan kolom, jumlah elemen array terdiri dari  $n$  buah baris dan  $m$  buah kolom. Array dua dimensi berbentuk seperti matriks atau tabel dimana indeks pertama menunjukkan baris dan indeks kedua adalah kolom. Berikut ilustrasi array 2 dimensi dengan matriks 3x4.

|         | Kolom 0 | Kolom 1 | Kolom 2 | Kolom 3 |
|---------|---------|---------|---------|---------|
| Baris 0 | (0,0)   | (0,1)   | (0,2)   | (0,3)   |
| Baris 1 | (1,0)   | (1,1)   | (1,2)   | (1,3)   |
| Baris 2 | (2,0)   | (2,1)   | (2,2)   | (2,3)   |

Pada gambar diatas terlihat jelas bahwa matriks dua dimensi tidak hanya membaca baris melainkan juga dengan kolom. Untuk pengisian model matriks seperti ini dapat dilakukan dengan 3 (tiga) langkah array satu dimensi. Hanya saja pada kolom index terdiri dari 2 yaitu baris dan kolom.

Tipe\_data nama\_var\_array [baris] [kolom] = {{elemen Baris 1 },...,{{elemen baris 3}}};

contoh pengisian pada code program adalah sebagai berikut:

```
int angka[3][4] ={{11,12,13,14},{15,16,17,18},{19,20,21,22}};
```

Atau bisa juga dengan menggunakan pengisian penugasan

```

int angka[3][4];
int main(){
    angka[0][0] = 11;
    angka[0][1] = 12;
    angka[0][2] = 13;
    angka[0][3] = 14;
    angka[1][0] = 15;
    angka[1][1] = 16;
    angka[1][2] = 17;
    angka[1][3] = 18;
    angka[2][0] = 19;
    angka[2][1] = 20;
    angka[2][2] = 21;
    angka[2][3] = 22;
}

```

Dari code program nilai elemen array diatas, kita dapat membuat laporan berbentuk matriks 3x4 dengan bentuk seperti berikut:

|    |    |    |    |
|----|----|----|----|
| 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 |

Code lengkap pembuatan array dua dimensi dengan menggunakan pengisian variabel adalah sebagai berikut:

```

1 #include <stdio.h>
2 int i,j; //baris 1      baris 2      baris 3
3 int angka[3][4] = {{11,12,13,14},{15,16,17,18},{19,20,21,22}};
4 int main(){
5     for(i=0;i<3;i++){
6         for(j=0;j<4;j++){
7             printf("%d ", angka[i][j]); // cetak array perbaris
8         }
9         printf("\n"); //membuat baris baru
10    }
11 }
12

```

Jika dilihat pada gambar diatas ada 2 (dua) kali melakukan perulangan, dimana perulangan pertama baris 5 yaitu perulangan `for(i=0;i<3;i++)` untuk mencetak baris sedangkan perulangan kedua adalah perulangan `for(j=0;j<4;j++)` mencetak kolom. Proses percetakan baris dan kolom dimulai dari baris 0 kolom 0 atau [0][0] sampai baris 0 kolom 3 [0][3]. Selanjutkan dengan membuat baris baru seperti code baris 9. Kemudian dilanjutkan baris 1 kolom 0 atau [1][0] sampai baris 1 kolom 3 [1][3] dan seterusnya sampai selesai. Dari code diatas akan mencetak hasil sebagai berikut:

```

E:\DATAKU\Dosent\Struktur DAta\2017 2018\CODE\array2d.exe
11 12 13 14
15 16 17 18
19 20 21 22

-----
Process exited after 0.1323 seconds with return value 0
Press any key to continue . . .

```

## 1. Array 2 dimensi Angka matriks

| Prodi                 | 2014 | 2015 | 2016 | 2017 |
|-----------------------|------|------|------|------|
| Sistem Informasi      | 20   | 35   | 50   | 145  |
| Teknik Informatika    | 15   | 40   | 24   | 100  |
| Manajemen Informatika | 13   | 43   | 23   | 56   |

Bentuk tabel diatas dapat dituangkan pada array berdimensi dua. Bentuk pendefinisianya sebagai berikut: `int data_lulus[3][4];` dimana 3 menyatakan sebagai baris (mewakili prodi) dan 4 menyatakan jumlah kolom (mewakili tahun kelulusan).

Pendefinisian diatas dapat diuraikan menjadi index seperti:

|        |      |      |      |      |
|--------|------|------|------|------|
| SI → 0 | 20   | 35   | 50   | 145  |
| TI → 1 | 15   | 40   | 24   | 100  |
| MI → 2 | 13   | 43   | 23   | 56   |
|        | 0    | 1    | 2    | 3    |
|        | 2014 | 2015 | 2016 | 2017 |

Code program

```

#include <stdio.h>
int main(){
    int data_lulus[3][4];// array berdimensi 2
    int tahun, jurusan;
    data_lulus[0][0]=20;
    data_lulus[0][1]=35;
    data_lulus[0][2]=50;
    data_lulus[0][3]=145;
    data_lulus[1][0]=15;
    data_lulus[1][1]=40;
    data_lulus[1][2]=24;
    data_lulus[1][3]=100;
    data_lulus[2][0]=13;
    data_lulus[2][1]=43;
    data_lulus[2][2]=23;
    data_lulus[2][3]=56;
}

```

```

while(1)
{
printf("Jurusan (0=SI, 1=TI,2=MI) :");
scanf("%d", &jurusan);
if((jurusan==0) || (jurusan==1) || (jurusan==2));
break;
}
while(1)
{
printf("Tahun (2014-2017):");
scanf("%d", &tahun);
if((tahun>=2014) && (tahun<=2017))
{
tahun=tahun-2014; //konversi ke 0,1,2,3
break;
}
}
printf("Jumlah yang lulus= %d",data_lulus[jurusan][tahun]);

return 0;
}

```

## 2. Array 2 dimensi dengan Penjumlahan

Array 2 dimensi yang dimaksud disini adalah menjumlahkan dua buah array pada indeks yang sama. Pendefinisian memerlukan 3 buah variabel yaitu A, B dan C. dimana variabel A dan B adalah nilai input array sedangkan variabel C adalah hasil penjumlahannya. Ilustrasi penjumlahan dapat dilihat pada tabel berikut:

|       |   |       |   |       |  |
|-------|---|-------|---|-------|--|
| $A =$ | $\begin{array}{ c c } \hline 3 & 4 \\ \hline 5 & 6 \\ \hline 7 & 8 \\ \hline \end{array}$ | $B =$ | $\begin{array}{ c c } \hline 6 & 8 \\ \hline 3 & 1 \\ \hline 4 & 5 \\ \hline \end{array}$ | $C =$ | $\begin{array}{ c c } \hline 9 & 12 \\ \hline 8 & 7 \\ \hline 11 & 13 \\ \hline \end{array}$ |
|-------|---|-------|---|-------|--|

Tabel diatas menunjukan adalah C merupakan hasil penjumlahan A + B, seperti pada matrik  $A[0,0] = 3$  dan  $B[0,0] = 6$  sehingga nilai  $C[0,0] = 9$ . Demikian sampai seterusnya.

```

#include <stdio.h>
int main(){
//definisi array 2dimensi
typedef int matrik32[3][2];
//deklarasi array A,B,C
matrik32 A,B,C;
int j,k;

```

```

//mengisi elemen array A
    for(j=0;j<3;j++)
    {   for(k=0;k<2;k++)
    {
        printf("A[%d] [%d] = ",j,k );
        scanf("%d", &A[j][k]);
    }
    } printf("\n");
//mengisi elemen array B
    for(j=0;j<3;j++)
    {   for(k=0;k<2;k++)
    {
        printf("B[%d] [%d] = ",j,k );
        scanf("%d", &B[j][k]);
    }
    } printf("\n");

//menjumlahkan array A dan B
    for(j=0;j<3;j++)
    {   for(k=0;k<2;k++)
    {
        C[j][k]=A[j][k] + B[j][k];
    }
    } //menampilkan hasil penjumlahan array

    for(j=0;j<3;j++)
    {   for(k=0;k<2;k++)
    {
        printf("C[%d] [%d] = %d \n",j,k, C[j][k] );
    }
    }
return 0;
}

```

```

A[0][0] = 3
A[0][1] = 5
A[1][0] = 6
A[1][1] = 2
A[2][0] = 4
A[2][1] = 6

B[0][0] = 2
B[0][1] = 3
B[1][0] = 5
B[1][1] = 7
B[2][0] = 8
B[2][1] = 9

C[0][0] = 5
C[0][1] = 8
C[1][0] = 11
C[1][1] = 9
C[2][0] = 12
C[2][1] = 15

```

## BAB 3

### STRUCTURE

Struktur merupakan kumpulan elemen data yang digabungkan menjadi satu kesatuan. Struktur memiliki persamaan dengan array yaitu alokasi memori untuk elemen-elemennya sudah ditentukan sebelum program dijalankan. Perbedaan keduanya terletak pada penggunaan tipe data, dimana array adalah struktur data yang tipe data memiliki elemen harus sama dan diakses melalui index sedangkan struktur adalah struktur data yang tipe data memiliki elemen tidak harus sama atau berbeda, dan elemen tersebut diakses melalui identifier atau nama variabel. Masing-masing elemen data dinamakan field atau elemen struktur. Field bisa memiliki bentuk tipe data yang sama ataupun berbeda, meskipun field dalam satu kesatuan tetapi dapat diakses secara individu.

Deklarasi struktur terdiri dari 2 (dua), program penulisan code struktur dapat dilihat pada struktur code dibawah ini:

```
struct nama_struktur // nama struktur, kata struct harus ada
{
    type1 element1;
    type2 element2; anggota / elemen dari struktur
    type3 element3;
    ..
} nama_object; // identifier yang digunakan untuk
pemanggilan struktur

-----atau-----
struct nama_struktur
{
    type1 element1;
    type2 element2;
    type3 element3;
    ..
} ;
struct nama_struktur nama_object;
```

Berikut code program penulisan struktur tipe data yang sama:

```
struct tanggal {
    Int tanggal;
    Int bulan;
    Int tahun;
}tgl;

-----atau-----
struct tanggal
{
    Int tanggal, bulan, tahun;
}tgl;
```

Berikut code program penulisan struktur tipe data yang berbeda:

```
struct mahasiswa {  
    char nim[10];  
    char nama[50];  
    float ipk;  
}mhs;
```

Contoh program penginputan data mahasiswa dengan menggunakan struktur:

```
#include <stdio.h>  
  
int main(){  
    struct mahasiswa {  
        char nim[10];  
        char nama [50];  
        float ipk;  
    }mhs;  
  
    printf("masukan nim anda : ");  
    scanf("%s", mhs.nim);  
    printf("masukan nama anda : ");  
    scanf("%s", mhs.nama);  
    printf("masukan IPK anda : ");  
    scanf("%f", &mhs.ipk);  
  
    printf("\nHASIL INPUTAN ADALAH \n" );  
    printf("nim      :%s \n", mhs.nim);  
    printf("nama     :%s \n", mhs.nama);  
    printf("nama     :%f \n", mhs.ipk);  
    return 0;  
}
```

### Output program

```
E:\DATAKU\Dosent\Matkulah\Struktur DAta\2017 2018\CODE\struck\struck.exe  
masukan nim anda : a001  
masukan nama anda : Andri  
masukan IPK anda : 2.75  
  
HASIL INPUTAN ADALAH  
nim      :a001  
nama     :Andri  
IPK      :2.750000
```

Baris code program diatas menunjukkan penggunaan 1 (satu) struktur. Selain dari contoh baris code tersebut kita juga dapat membuat *struck* dalam *struck* dengan menggunakan code program berikut:

```

struct data_tgl{
    int tgl, bulan, tahun;
};

struct mahasiswa {
    char nim[10];
    char nama [50];
    struct data_tgl tgl;
}mhs;

```

Ilustrasi struck dalam struck dengan mencontohkan data mahasiswa, dimana struck induk adalah adalah **struck mahasiswa** dengan struck tambahan adalah **struck data\_tgl**. Code lengkap programnya:

```

#include <stdio.h>

int main(){
    struct data_tgl{
        int tgl, bulan, tahun;
    };
    struct mahasiswa {
        char nim[10];
        char nama [50];
        struct data_tgl tgl;
    }mhs;

    printf("masukan nim anda : ");
    scanf("%s", mhs.nim);
    printf("masukan nama anda : ");
    scanf("%s", mhs.nama);

    printf("masukan tanggal lahir anda : ");
    scanf("%d", &mhs.tgl.tgl);
    printf("masukan bulan lahir anda : ");
    scanf("%d", &mhs.tgl.bulan);
    printf("masukan tahun lahir anda : ");
    scanf("%d", &mhs.tgl.tahun);

    printf("\nHASIL INPUTAN ADALAH \n" );
    printf("nim :%s \n", mhs.nim);
    printf("nama :%s \n", mhs.nama);
    printf("Tanggal Lahir :%d-%d-%d \n", mhs.tgl.tgl,
mhs.tgl.bulan,mhs.tgl.tahun );
    return 0;
}

```

## Enumerasi

Enumerasi adalah tipe data yang nilainya terbatas pada nilai nilai yang telah didefinisikan saja. Tipe Enumerasi digunakan untuk membentuk tipe data yang nilainya bersifat pasti. Misalnya untuk mendefinisikan tipe jenis kelamin, nama hari, warna primer dan sebagainya. Kita tahu bahwa jenis kelamin hanya terdiri dari pria dan wanita saja, maka jenis kelamin dapat kita bentuk ke dalam tipe enumerasi.

```
#include <stdio.h>
enum hari {minggu=1, senin, selasa, rabu, kamis, jumat, sabtu}; // deklarasi dan isi data ditentukan sendiri

int main()
{
    hari day; // inisialisasi ke tipe data hari
    day=senin;

    printf("%d", day);
    return 0;
}
```

Contoh lain code enumerasi

```
#include <stdio.h>
main(){
    struct Data // deklarasi struktur dengan nama Data
    {
        int Mid, Uas;
        int Nilai_Akhir; } ;

    Data Almin = {80,90}, Ardi = { 70, 60};

    Almin.Nilai_Akhir = Almin.Mid* Almin.Uas / 2;

    Ardi.Nilai_Akhir = Ardi.Mid*Ardi.Uas/2;
    printf("Nilai Almin : %d \n", Almin.Nilai_Akhir);
    printf("Nilai Adi : %d \n", Ardi.Nilai_Akhir);
}
```

## Latihan

Buatlah program untuk menghitung spp mahasiswa menggunakan struktur, jika diketahui :

a. D3

- spp tetap Rp 500.000
- spp var Rp 25.000/sks

b. S1

- spp tetap Rp 750.000
- spp var Rp 50.000/sks

```

#include <stdio.h>

struct mhs
{ char nama[20],nim[10],jurusan[2];
int sks,program; };
struct mhs bayar;
main ()
{
int bts,var,tetap;
//input data
printf("Nama mhs      = "); scanf("%s", bayar.nama);
printf("NIM           = "); scanf("%s", bayar.nim);
printf("Jurusan[SI,TI,MI] = "); scanf("%s", bayar.jurusan);
input:
printf("Program[1=D3,2=S1]= "); scanf("%d", &bayar.program);

if (bayar.program < 0 || bayar.program > 2)
{printf("Program tidak sesuai\n");
goto input;}
printf("Jumlah sks      =" ); scanf("%d", &bayar.sks);
if (bayar.program==1)
{tetap=500000;
var=bayar.sks*25000;}
else if (bayar.program==2)
{tetap=750000;
var=bayar.sks*50000;}

//output data
printf("\n-----\n");
printf("HASIL");
printf("\n-----\n");

printf("Nama Mahasiswa = %s \n",bayar.nama );
printf("Nama NIM       = %s \n",bayar.nim );
printf("Jurusan        = %s \n",bayar.jurusan );
printf("Program        = %d \n",bayar.program );
printf("SKS            = %d \n",bayar.sks );
printf("SPP Tetap     = %d \n",tetap );
printf("SPP Tetap     = %d \n",var);

return 0;
}

```

## BAB 4

# SORTING

Sorting merupakan proses mengatur sekumpulan objek menurut aturan atau susunan tertentu. Urutan objek tersebut terdiri dari *ascending* (urutan data dari kecil ke data lebih besar) dan *descending* (urutan data dari besar ke data terkecil). Algoritma-algoritma pengurutan yang diklasifikasikan sebagai *exchange sort* melakukan pembandingan antar data, dan melakukan pertukaran apabila urutan yang didapat belum sesuai. Banyak algoritma pengurutan yang ada, akan tetapi disini kita pelajari 3 metode yaitu:

- Bubble sort (gelembung)
- Selection sort (maksimum/minimun)
- Insertion sort (sisip)

### A. Buble sort (Gelembung)

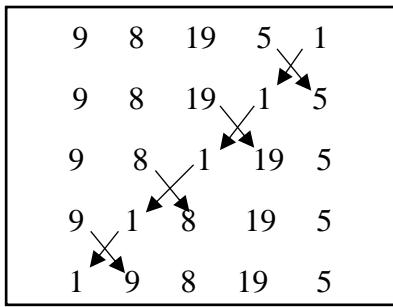
Pengurutan metode bubble sort diinspirasi oleh gelembung sabun yang ada di permukaan air. Karena berat jenis gelembung sabun lebih ringan daripada berat jenis air maka gelembung sabun akan selalu mengapung. Prinsip pengapungan ini juga dipakai pada pengurutan gelembung. Elemen yang berharga paling kecil “diapungkan”, artinya diangkat ke atas (atau ke ujung paling kiri) melalui pertukaran. Keuntungan dari algoritma sorting ini adalah karena paling mudah diaplikasikan.

- 1) Konsep algoritma bubble sort
  - a. Algoritma dimulai dari elemen paling awal.
  - b. 2 buah elemen pertama dari list dibandingkan. Jika elemen pertama lebih besar dari elemen kedua atau sebaliknya (urut secara *ascending* atau *descending*) kemudian dilakukan pertukaran.
  - c. Langkah 2 dan 3 dilakukan lagi terhadap elemen kedua dan ketiga, seterusnya sampai ke ujung elemen
  - d. Bila sudah sampai ke ujung dilakukan lagi ke awal sampai tidak ada terjadi lagi pertukaran elemen.
  - e. Bila tidak ada pertukaran elemen lagi, maka list elemen sudah terurut.
  - f. Setiap pasangan data:  $x[j]$  dengan  $x[j+1]$ , untuk semua  $j=1,\dots,n-1$  harus memenuhi keterurutan, yaitu untuk pengurutan:
    - o Ascending :  $x[j] < x[j+1]$
    - o Descending :  $x[j] > x[j+1]$

2) Implementasi bubble sort secara Ascending

Berikut adalah ilustrasi Algoritma bubble sort

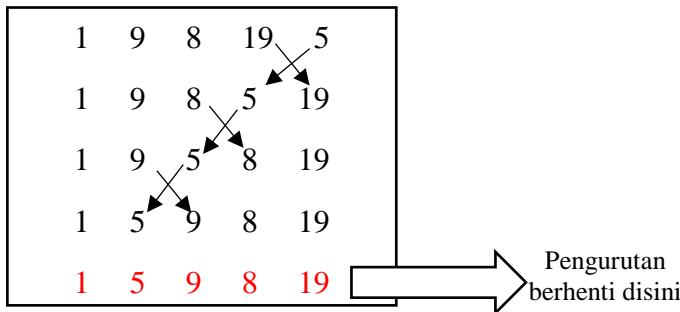
**Proses 1**



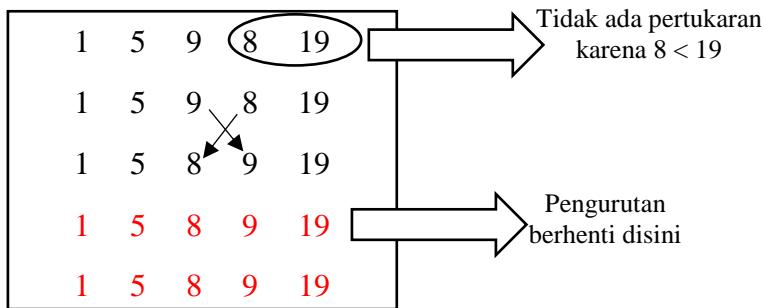
Gambar 3.1 Ilustrasi Algoritma Bubble sort

- Pada Gambar 3.1 pengecekan dimulai dari data yang paling akhir, kemudian di bandingkan dengan data di depannya, jika data di depannya lebih besar maka akan ditukar
- Pada proses kedua, proses pertama diulangi dan pengecekan dilakukan sampai dengan data ke-2 karena data pertama pasti sudah paling kecil.
- Proses ketiga dilakukan dengan cara yang sama dengan proses pertama atau proses kedua sampai data sudah dalam kondisi terurut. Proses iterasi maksimal dilakukan sampai dengan  $n-1$ , dimana  $n$  adalah jumlah data yang akan diurutkan

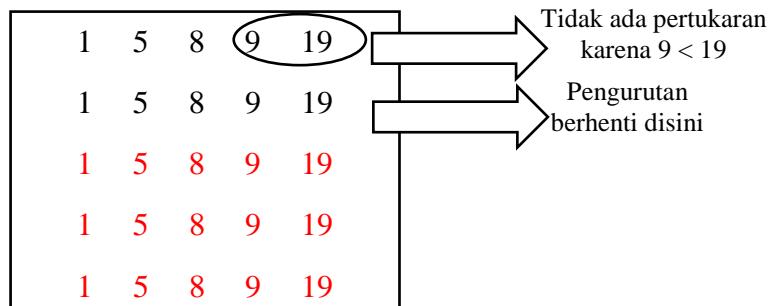
**Proses 2**



**Proses 3**



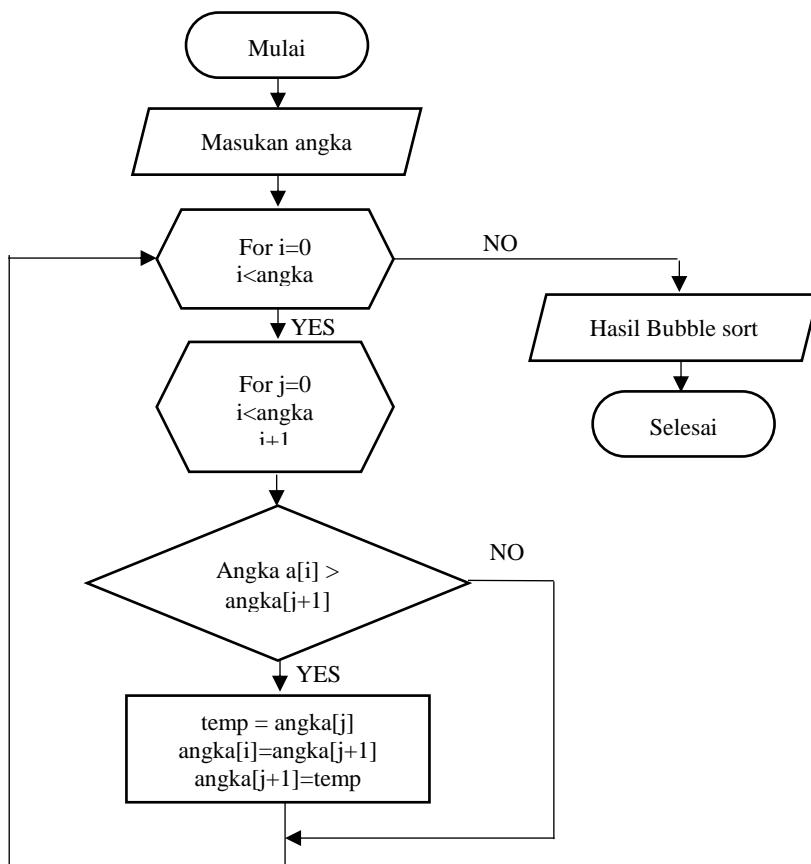
#### Proses 4



Gambar 3.2 Ilustrasi Algoritma Bubble sort

Jika kita melihat pada Gambar Proses 3 (tiga) sebenarnya proses pengurutan data sudah benar. Namun metode pengurutan data selalu membandingkan keseluruhan data. Pada ilustrasi diatas data yang digunakan adalah 5 sehingga jumlah prosesnya 4 kali melakukan perbandingan.

### 3) Algoritma Bubble sort



- 1) Mulai.
- 2) Memasukan data.
- 3) Membandingkan setiap pasang elemen data yang berdekatan dari awal, misal data pertama dengan data kedua.

- 4) Jika nilai elemen yang pertama lebih besar dari elemen yang kedua maka elemen tersebut di tukar posisi hingga nilai elemen paling awal harus lebih kecil dari elemen kedua.
- 5) Membandingkan elemen kedua dengan elemen ketiga, apakah lebih kecil atau masih lebih besar jika lebih kecil lakukan penukaran dengan elemen ketiga
- 6) Mengulangi perbandingan di atas hingga kondisi elemen benar benar urut dari awal sampai akhir.
- 7) Selesai.

4) Procedure bubble sort

```

2  #include <stdio.h>
3  int total,data[10];
4
5  void input(){
6      printf("masukan nilai yang di sort = ");scanf("%d",&total);
7      for(int a=0;a<total;a++){
8          printf("masukkan nilai pada INDEX ke %d = ",a+1);scanf("%d",&data[a]);
9      }
10 }
11 void sort(){
12     int temp;
13     for(int a=0;a<total-1;a++){
14         for(int b=0;b<total-1;b++){
15             if(data[b]>data[b+1]){
16                 temp=data[b+1];
17                 data[b+1]=data[b];
18                 data[b]=temp;
19             }
20         }
21     }
22 }
23 void view(){
24     for(int a=0;a<total;a++){
25         printf("%d ",data[a]);
26     }
27     printf("\n");
28 }
29 int main(){
30     input();
31     printf("sebelum di- sorting\n");
32     view();
33     sort();
34     printf("sesudah di- sorting\n");
35     view();
36 }
```

## B. Selection sort (Maksimum/Minimum)

### 1. Konsep algoritma selection sort

Metode selection sort merupakan perbaikan dari metode bubble sort dengan mengurangi jumlah perbandingan. Selection sort metode pengurutan dengan *mencari nilai data terkecil* dimulai dari data diposisi 0 hingga diposisi N-1. Jika terdapat N data dan data terkoleksi dari urutan 0 sampai dengan N-1 maka algoritma pengurutan dengan metode selection sort adalah sebagai berikut :

- Cari data terkecil dalam interval  $a = 0$  sampai dengan  $a = N-1$

- Jika pada posisi data ditemukan data yang terkecil, tukarkan data diposisi pos dengan data di posisi b.
- Ulangi langkah 1 dan 2 dengan  $b = b + i$  sampai dengan  $b = N-1$ , dan seterusnya sampai  $b = N - 1$ .

Bila diketahui data awal berupa: 85 63 24 45 17 31 96 50, maka langkah per langkah pengurutan dengan metode selection sort adalah sebagai berikut:

## 2. Implementasi algoritma selection sort

Implementasi algoritma selection sort, misalkan data yang akan disortir adalah sebagai berikut:

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 85 | 63 | 24 | 45 | 17 | 31 | 96 | 50 |
|----|----|----|----|----|----|----|----|

Tabel 3.1. Langkah demi langkah pengurutan dengan metode Selection Sort.

| 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | Urutan                              |
|----|----|----|----|----|----|----|----|-------------------------------------|
| 85 | 63 | 24 | 45 | 17 | 31 | 96 | 50 | Data Awal                           |
| 17 | 63 | 24 | 45 | 85 | 31 | 96 | 50 | Tukarkan data ke 1 dengan data ke 5 |
| 17 | 24 | 63 | 45 | 85 | 31 | 96 | 50 | Tukarkan data ke 2 dengan data ke 3 |
| 17 | 24 | 31 | 45 | 85 | 63 | 96 | 50 | Tukarkan data ke 3 dengan data ke 6 |
| 17 | 24 | 31 | 45 | 85 | 63 | 96 | 50 | Data ke 4 tidak pertukaran          |
| 17 | 24 | 31 | 45 | 50 | 63 | 96 | 85 | Tukarkan data ke 5 dengan data ke 8 |
| 17 | 24 | 31 | 45 | 50 | 63 | 96 | 85 | Data ke 6 tidak pertukaran          |
| 17 | 24 | 31 | 45 | 50 | 63 | 85 | 96 | Tukarkan data ke 7 dengan data ke 8 |
| 17 | 24 | 31 | 45 | 50 | 63 | 85 | 96 | Data selesai                        |

## 3. Procedure bubble sort

```
//selection sort
#include <stdio.h>
int total,data[10];
void input(){
printf("****SELECTION SORT****\n");
printf("input Jumlah yang di sort = ");scanf("%d",&total);
    for(int a=0;a<total;a++){
        printf("masukkan nilai pada INDEX ke %d = ",a+1);scanf("%d",&data[a]);
    }
}
void sort(){
int temp;
    for(int a=0;a<total-1;a++){
        for(int b=a+1;b<total;b++){
            if(data[a]>data[b]){
                temp=data[b];
                data[b]=data[a];
                data[a]=temp;
            }
        }
    }
}
```

```

void view(){
    for(int a=0;a<total;a++){
        printf("%d ",data[a]);
    }
    printf("\n");
}

int main(){
input();
    printf("sebelum di- sorting\n");
view();
sort();
    printf("sesudah di- sorting\n");
}

```

### C. Insertion sort (sisip)

#### 1. Konsep algoritma insertion sort

Prinsip dasar *Insertion* adalah secara berulang-ulang menyisipkan/ memasukan setiap elemen ke dalam posisinya/tempatnya yang benar. Mirip dengan cara orang **mengurutkan** kartu, selembar demi selembar kartu diambil dan **disisipkan** (*insert*) ke tempat yang seharusnya. Pengurutan dimulai dari data ke-2 sampai dengan data terakhir, jika ditemukan data yang **lebih kecil**, maka akan ditempatkan (**diinsert**) diposisi yang seharusnya. Pada penyisipan elemen, maka elemen-elemen lain akan bergeser ke belakang.

#### 2. Implementasi Insertion sort

- 1) Kondisi awal:
  - Unsorted list = data
  - Sorted list = kosong
- 2) Ambil sembarang elemen dari *unsorted list*, sisipkan (*insert*) pada posisi yang benar dalam *sorted list*. Lakukan terus sampai *unsorted list* habis.

Langkah 1

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 85 | 63 | 24 | 45 | 17 | 31 | 96 | 50 |
|----|----|----|----|----|----|----|----|

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 36 | 85 | 24 | 45 | 17 | 31 | 96 | 50 |
| 24 | 36 | 85 | 45 | 17 | 31 | 96 | 50 |
| 24 | 36 | 45 | 85 | 17 | 31 | 96 | 50 |

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 17 | 24 | 36 | 45 | 85 | 31 | 96 | 50 |
|----|----|----|----|----|----|----|----|

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 17 | 24 | 36 | 45 | 85 | 31 | 96 | 50 |
|----|----|----|----|----|----|----|----|

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 17 | 24 | 31 | 36 | 45 | 85 | 96 | 50 |
|----|----|----|----|----|----|----|----|

Langkah 2

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 17 | 24 | 31 | 36 | 45 | 85 | 96 | 50 |
|----|----|----|----|----|----|----|----|

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 17 | 24 | 31 | 36 | 45 | 85 | 50 | 95 |
| 17 | 24 | 31 | 36 | 45 | 50 | 85 | 95 |

### 3. Procedure Insertion sort

```
//Insertion Sort
#include <stdio.h>
int total,data[10];
void input(){
printf("**** INSERTION SORT ****");scanf("%d",&total);
printf("Jumlah yang diproses= ");scanf("%d",&total);
for(int a=0;a<total;a++){
printf("masukkan nilai pada INDEX ke %d = ",a+1);scanf("%d",&data[a]);
}
}

void sort(){
int temp,key,i;
for(int a=0;a<total;a++){
key=data[a];
i=a-1;

while(i>=0 && data[i]>key){
    data[i+1]=data[i];
    i=i-1;
    data[i+1]=key;
}
printf("%d \n",data[a]);
} }

void view(){
for(int a=0;a<total;a++){
printf("%d ",data[a]);
}
printf("\n");
}

int main(){
input();
printf("sebelum di- sorting\n");
view();
sort();
printf("sesudah di- sorting\n");
view();
}
```

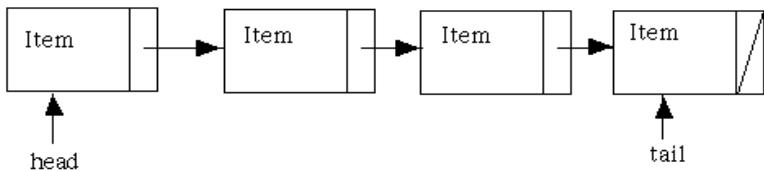
## BAB 5

### LINKED LIST

Linked List atau dikenal juga dengan sebutan senarai berantai adalah struktur data yang terdiri dari urutan record data dimana setiap record memiliki field yang menyimpan alamat/referensi dari record selanjutnya (dalam urutan). Elemen data yang dihubungkan dengan link pada Linked List disebut *Node*. Biasanya didalam suatu linked list, terdapat istilah *head* dan *tail*. *Head* adalah elemen yang berada pada posisi pertama dalam suatu linked list sedangkan *tail* adalah sebaliknya yaitu elemen yang berada paling belakang suatu linked list. Ada beberapa macam linked list yaitu single linked list, double linked list, circular linked list, dan multiple linked list.

#### 1. Single Linked List

Single linked list atau biasa disebut linked list terdiri dari elemen-elemen individu, dimana masing-masing dihubungkan dengan pointer tunggal. Masing-masing elemen terdiri dari dua bagian, yaitu sebuah data dan sebuah pointer yang disebut dengan pointer *next*. Pointer *next* pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut *head*, dan elemen terakhir dari suatu list disebut *tail*.



```
struct Mahasiswa{  
    char nim[12];  
    char nama[25];  
    int usia;  
    struct Mahasiswa *next;  
}*head,*tail;
```

#### 2. Double linked list

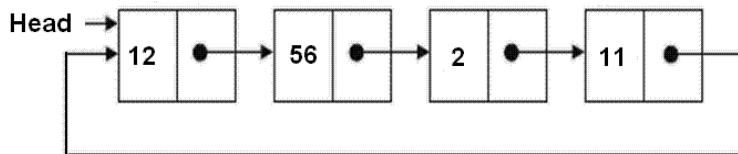
Double Linked List merupakan suatu linked list yang memiliki dua variabel pointer. Dimana pointer yang menunjuk ke node selanjutnya dan pointer yang menunjuk ke node sebelumnya. Setiap *head* dan *tail*nya juga menunjuk ke NULL.

```
struct Mahasiswa{  
    char nim[12];  
    char nama[25];  
    int usia;  
    struct Mahasiswa *next,*prev;  
}*head,*tail;
```

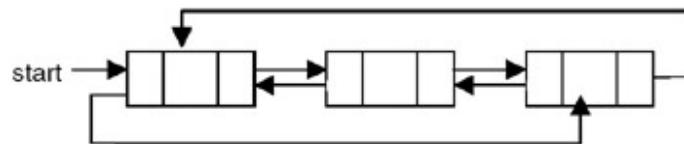
### 3. Circular linked list

Ada 2 (dua) jenis circular linked list yaitu circular single linked list dan circular double linked list. Cara kerja circular linked list dimana tail atau node terakhir menunjuk ke head atau node pertama. Dalam hal ini tidak ada pointer yang menunjuk NULL.

- Circular single linked list

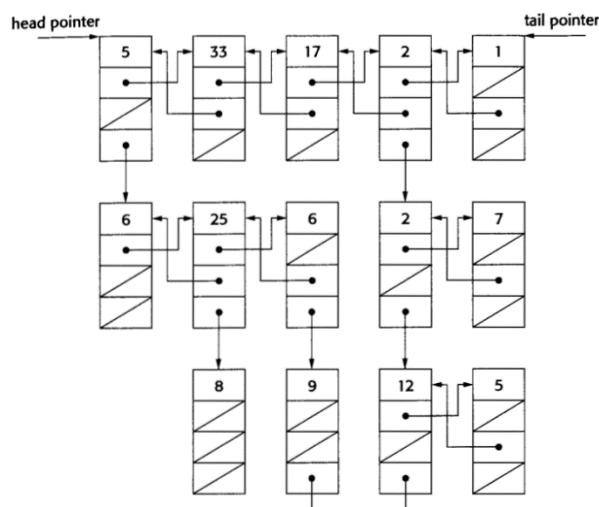


- Circular double linked list



### 4. Multiple linked list

Multiple Linked List memiliki link atau pointer lebih dari satu. Untuk multiple linked list yang memiliki dua link biasanya disebut sebagai *doublylinked* list (senarai berantai ganda). Senarai berantai ganda memiliki dua buah pointer yang biasanya masing-masing menunjuk ke simpul sebelumnya dan ke simpul sesudahnya. suatu linked list yang memiliki lebih dar 2 buat variabel pointer. contoh:



## A. Algoritma Linked list

Struktur data memiliki 2 penunjuk supaya bisa saling terhubung antara elemen yang satu dengan elemen yang lainnya. Dengan menggunakan struktur data sebagai berikut:

- a) Data
- b) pointer penunjuk elemen setelahnya (next)
- c) pointer penunjuk elemen sebelumnya (prev)

Langkah pembuatan program diperlukan beberapa library seperti:

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

Untuk merepresentasikan bisa seperti dibawah ini, dengan membuat variabel global pada fungsi TNode. Yang terdiri dari data, TNode, head, tail, depan dan belakang.

```
struct TNode{
    int data;
    TNode *next;
};TNode *head, *tail,*depan,*belakang;
```

Ada beberapa fungsi yang dibuat seperti fungsi ini untuk menulukan node atau simpul

Fungsi init

```
void init(){
    head=NULL;
    tail=NULL;
    depan=NULL;
    belakang=NULL;
}
```

Fungsi isEmpty

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

Untuk mengetahui apakah suatu *tail* dalam keadaan kosong adalah dengan membandingkan *tail* barus, jika kondisi benar maka node dalam posisi kosong, dan sebaliknya

Fungsi Tambah didepan

```
void insertDepan(int databaru)
{
    TNode *baru;
    baru =new TNode;
    baru->data=databaru;
    baru->next=NULL;
    if(isEmpty()==1)
    {
        head=tail=baru;
        tail->next=NULL;
    }else{
        baru->next=head;
        head=baru;
    }
    cout<<"Input berhasil";
}
```

Untuk menambah node pada bagian depan dengan menggunakan fungsi diatas, sedangkan untuk menambah node bagian belakang dengan menggunakan fungsi seperti berikut:

Fungsi tambah belakang

```
void insertBelakang(int data){
    TNode *baru;
    baru=new TNode;
    baru->data=data;
    baru->next=NULL;
    if(isEmpty() == 1){
        head=baru;
        tail=baru;
        tail->next=NULL;
    }else
    {
        tail->next=baru;
        tail=baru;
    }
    cout<<"\n Input berhasil"<<endl;
}
```

Selain ada penambahan node linked list juga bisa menyediakan penghapusan node. Dimana node yang akan dihapus dimulai dari belakang ataupun dari belakang. Fungsi penghapusan node didepan dengan menggunakan fungsi berikut:

Fungsi hapus depan

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

Fungsi hapus belakang

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

Dari kedua fungsi penghapusan data diatas ada perintah *isEmpty()*. Ini berfungsi mengecek keadaan data dalam node apabila terdapat data maka akan dihapus jika tidak maka akan muncul informasi “masih kosong”.

Fungsi yang paling terakhir adalah *tampil()*. Ini berfungsi untuk menampilkan data yang sudah dimasukan kedalam node.

Fungsi tampilkan data

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

## B. Procedure linked list

Code lengkap linked list

```
#include<iostream>
#include<cstdlib>
#include<conio.h>
using namespace std;

struct TNode{
    int data;
    TNode *next;
};TNode *head, *tail,*depan,*belakang;

void init(){
    head=NULL;
    tail=NULL;
    depan=NULL;
    belakang=NULL;
}
```

```

int isEmpty(){
    if(tail==NULL){
        return 1;
    }else{
        return 0;
    }
}
void insertDepan(int databaru)
{
    TNode *baru;
    baru =new TNode;
    baru->data=databaru;
    baru->next=NULL;
    if(isEmpty()==1)
    {
        head=tail=baru;
        tail->next=NULL;
    }else{
        baru->next=head;
        head=baru;
    }
    cout<<"Input berhasil";
}
void insertBelakang(int data){
    TNode *baru;
    baru=new TNode;
    baru->data=data;
    baru->next=NULL;
    if(isEmpty()==1){
        head=baru;
        tail=baru;
        tail->next=NULL;
    }else
    {
        tail->next=baru;
        tail=baru;
    }
    cout<<"\n Input berhasil"<<endl;
}
void hapusDepan(){
    TNode *hapus;
    int d;
    if(isEmpty()==0){
        if(depan!=belakang){
            hapus=depan;
            d=hapus->data;
            depan=depan->next;
            delete hapus;
        }else{
            d=belakang->data;
            depan=belakang=NULL;
        }
        cout<<d<<"Terhapus";
    }else cout<<"Masih kosong\n";
}

```

```

void hapusBelakang(){
    TNode *bantu,*hapus;
    int d;
    if(isEmpty()==0){
        bantu=depan;
        if(depan!=belakang){
            while(bantu->next!=belakang){
                bantu=bantu->next;
            }
            hapus=belakang;
            belakang=bantu;
            d=hapus->data;
            delete hapus;
            belakang->next=NULL;
        }else{
            d=belakang->data;
            depan=belakang=NULL;
        }
        cout<<d<<"Terhapus\n";
    }else cout<<"Masih kosong\n";
}
void tampil(){
    TNode *bantu;
    bantu=head;
    if(isEmpty()==0){
        cout<<"depan :";
        while(bantu!=NULL){
            cout<<bantu->data<<"->";
            bantu=bantu->next;
        }
    }else if(isEmpty()==0){
        cout<<"belakang :";
        while(bantu!=NULL){
            cout<<bantu->data<<"->";
            bantu=bantu->next;
        }
        cout<<"\n Data masih kosong"<<endl;
    }
}

```

```

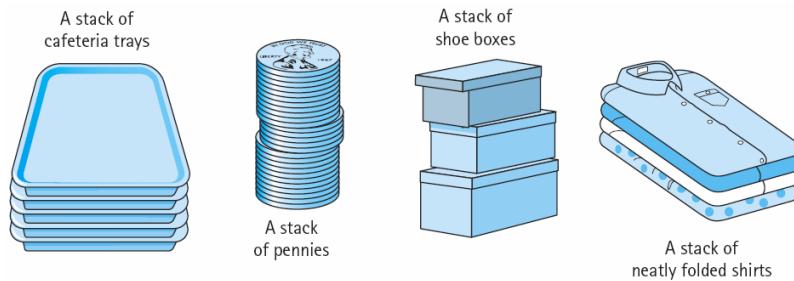
main(){
    int pil,data,databaru;
    init();
    do{
        system("cls");
        cout<<endl;
        cout<<"CONTOH PROGRAM LINKED LIST"<<endl;
        cout<<"-----"<<endl;
        cout<<"1. tambah depan"<<endl;
        cout<<"2. tambah belakang"<<endl;
        cout<<"3. hapus depan"<<endl;
        cout<<"4. hapus belakang"<<endl;
        cout<<"5. tampilan data"<<endl;
        cout<<"0. keluar"<<endl;
        cout<<"-----"<<endl;
        cout<<"masukkan pilihan :";cin>>pil;
        switch(pil){
            case 1: system("cls");
                cout<<"\ntambah Depan"<<endl;
                cout<<"-----"<<endl;
                cout<<"masukkan data : ";cin>>databaru;
                insertDepan(databaru);
                break;
            }
            case 2: system("cls");
                cout<<"\nTambah Belakang"<<endl;
                cout<<"-----"<<endl;
                cout<<"masukkan data : ";cin>>data;
                insertBelakang(data);
                break;
            }
            case 3: system("cls");
                cout<<"\ndata depan terhapus"<<endl;
                hapusDepan();
                break;
            }
            case 4: system("cls");
                cout<<"\nData Belakang terhapus"<<endl;
                hapusBelakang();
                break;
            }
            case 5: system("cls");
                cout<<"Tampilan data"<<endl;
                cout<<"-----"<<endl;
                tampil();
                break;
            }
            case 0 : system("cls");
                return 0;
                break;
            }default:
                system ("cls");
                cout<<"pilihan tidak tersedia"<<endl;
            }
        }getch();
    }while(pil!=7);
}

```

## BAB 6

### STACK

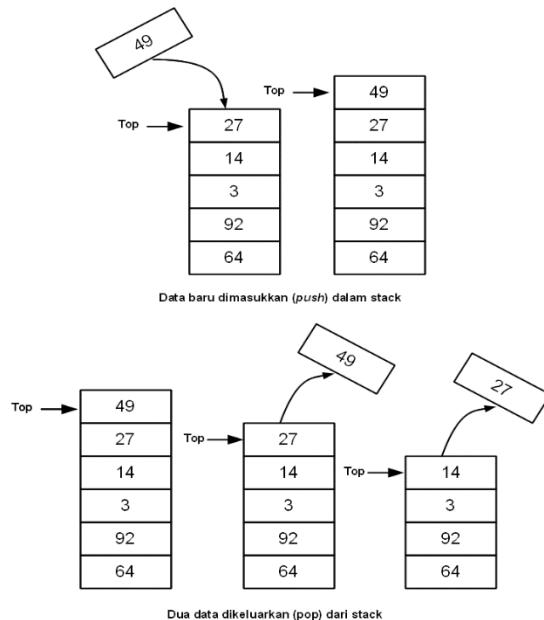
Stack adalah struktur data yang menggunakan konsep LIFO(Last In First Out). Dimana kumpulan data yang diletakkan di atas tumpukan data yang lain. Dengan demikian, elemen terakhir yang disimpan dalam stack menjadi elemen pertama yang diambil. Dalam proses komputasi, untuk meletakkan sebuah elemen pada bagian atas dari stack, maka dilakukan operasi push.



Gambar 6.1 Ilustrasi stack

Operasi-operasi stack untuk mengatur data:

- **Push** : untuk menambah item pada stack pada tumpukan paling atas
- **Pop** : untuk mengambil item pada stack pada tumpukan paling atas
- **Clear** : untuk mengosongkan stack
- **IsEmpty** : untuk mengecek apakah stack sudah kosong
- **IsFull** : untuk mengecek apakah stack sudah penuh



Jika kita melihat dari gambar diatas stack dapat dianalogikan sebuah wadah yang diisi benda hanya satu pintu keluar dan masuk.

## A. Pendeklarasian stack

Proses pendeklarasian *stack* adalah proses pembuatan struktur *stack* dalam memori. Suatu *stack* memiliki beberapa bagian yaitu :

- a. **top** yang menunjuk posisi data terakhir (TOP)
- b. **data** yang berisi data yang ada dalam *stack*. Bagian ini lah yang berbentuk array.

```
//pendeklarasian struct
struct STACK
{
    int top;
    float data[];
};

float dta;
```

### 1) Inisialisasi

Inisialisasi *stack* adalah proses pembuatan suatu *stack* kosong. Proses inisialisasi untuk *stack* yang menggunakan array adalah dengan mengisi nilai field top dengan 0 (nol) Implementasinya dalam bahasa Pascal adalah sebagai berikut

```
struct STACK stackbaru;
```

## B. Procedure code stack

```
//header file
#include <iostream>
#include <conio.h>
#define maxstack 5
using namespace std; //untuk melegalkan header iostream

//pendeklarasian struct
struct STACK
{
    int top;
    float data[4];
};

float dta;

/*struct yang telah dibuat (STACK) dijadikan suatu Tipe data, dimana disebut tipe data
abstrak*/
struct STACK stackbaru;
```

```

//fungsi boolean untuk mengetahui apakah stack penuh
bool isfull()
{
    if(stackbaru.top == maxstack)
        return true;
    else
        return false;
}
//fungsi boolean untuk mengetahui apakah stack kosong
bool isempty()
{
    if(stackbaru.top == -1)
        return true;
    else
        return false;
}
//fungsi untuk menambahkan data pada stack
void push(float dta)
{
    if(isfull() == true) /* panggil fungsi isempty(), jika kondisi benar pernyataan dijalankan*/
    {
        puts("Maaf, stack penuh");
    }
    else{
        stackbaru.top++;
        stackbaru.data[stackbaru.top]=dta;
    }
}

//fungsi untuk mengambil data pada stack
void pop()
{
    if(isempty() == true) // panggil fungsi isempty(), jika kondisi benar pernyataan dijalankan
    {
        cout<<"Data telah kosong!";
    }
    else
    {
        cout<<"Data yang terambil : " <<stackbaru.data[stackbaru.top]<<endl;
        stackbaru.top--;
    }
}

//fungsi untuk mencetak data pada stack
void print()
{
    printf("\nData yang terdapat dalam stack : \n");
    printf("-----\n");
    for(int i=0; i<=stackbaru.top; i++)
    {
        cout<<stackbaru.data[i]<<" ";
    }
}

//fungsi untuk membersihkan data dalam stack
void clear()
{
    stackbaru.top = -1;
    printf("\nSekarang stack kosong");
}

```

```

//fungsi utama
int main()
{
    stackbaru.top = -1;
    //pendeklarasian variabel
    char menu;
    char ulang;
    //perulangan dengan do-while
    do
    {
        system("cls");
        printf("\t PROGRAM STACK\n");
        printf("\t=====\n");
        printf("Menu : ");
        puts("\n1. Pop stack");
        puts("2. Push stack");
        puts("3. Cetak");
        puts("4. Bersihkan stack");
        puts("5. Exit");

        cout<<"Menu pilihan Anda : ";
        cin>>menu;

        if(menu == '1')
        {
            pop(); //panggil fungsi pop()
            ulang = 'y';
            getch();
        }
        else if(menu == '2')
        {
            cout<<"\nTambah Data";
            cout<<"\n-----";
            cout<<"\nData yang akan disimpan di stack : ";
            cin>>dta;
            push(dta); /*panggil fungsi push(dta)--dta sesuai dengan data yg yg diinput*/
            ulang = 'y';
        }
        else if(menu == '3')
        {
            print(); /*panggil fungsi untuk mencetak data dalam stack*/
            cout<<"\n\nUlang ? (y/t)";
            cin>>ulang;
        }
        else if(menu == '4')
        {
            clear(); //panggil fungsi untuk membersihkan stack
            cout<<"\n\nUlang ? (y/t)";
            cin>>ulang;
        }
        else if(menu == '5')
        {
            exit(0); //keluar dari program
        }
    }while(ulang == 'Y' || ulang == 'y'); /*akan selalu diulang ketika ulang == 'y' || ulang='Y'*/
}

```

### C. Penjelasan Code Program

- a) Pertama kali dibuat sebuah struct ‘struct STACK’, kemudian struct stack dijadikan menjadi sebuah tipe data dari variabel stackbaru. Dan beberapa model fungsi yang dibuat diantaranya :

```
void clear()
void print()
void pop()
void push(float dta)
bool isempty()
bool isfull()
```

- b) Pada fungsi main membuat deklarasi variabel dengan memasukan perintah perulangan **do-while** dengan kondisi **ketika ulang == ‘y’ || ulang ‘Y’**. Perulangan ini akan ditampilkan beberapa menu pilihan yang dapat dipilih oleh user :

```
printf("Menu : ");
puts("\n1. Pop stack");
puts("2. Push stack");
puts("3. Cetak");
puts("4. Bersihkan stack");
puts("5. Exit");
/*note = puts(put string), kegunaannya sama dengan printf ataupun cout, hanya
saja yang dicetak hanyalah berupa STRING.*/
```

- c) Jika user memilih 1 atau *menu==1* maka program akan menjalankan seluruh pernyataan yang ada pada kondisi pertama, yaitu pemanggilan fungsi pop();

#### Fungsi Pop

```
//fungsi untuk mengambil data pada stack
void pop()
{
    if(isempty() == true) // panggil fungsi isempty(), jika kondisi benar
    pernyataan dijalankan
    {
        cout<<"Data telah kosong!";
    }
    else
    {
        cout<<"Data yang terambil : "
        <<stackbaru.data[stackbaru.top]<<endl;
        stackbaru.top--;
    }
}
```

Pertama kali yang akan dilakukan program adalah mengecek apakah stack dalam keadaan kosong, jika benar maka akan tercetak string pada layar “Data telah kosong!”. Jika bernilai false

maka data pada posisi teratas akan diambil, dan kemudian nilai stackbaru.top didecrement sehingga posisi teratas pada stack berganti dengan data dibawah top sebelumnya.  
ulang = 'y';

Setelah fungsi pop() dijalankan, selanjutnya variabel ulang disetel dengan Nilai ‘y’ Sehingga setelah memilih menu pop() program akan secara otomatis mengulangi prosesnya. Perintah *getch()* yaitu menghentikan program sementara sampai adanya input keyboard.  
*getch();*

- d) Jika memilih nilai 2 atau menu == ‘2’, maka pernyataan yang berada di dalam kondisi ke dua akan dijalankan. Saat masuk dalam pernyataan pada kondisi kedua maka akan tercetak String pada layar dan kemudian user diminta untuk mengisikan data yang akan ditambahkan dalam tumpukan stack.

```
cout<<"\nTambah Data";
cout<<"-----";
cout<<"\nData yang akan disimpan di stack : ";
cin>>dta;
```

Data yang telah diinputkan disimpan dalam memori, dan digunakan untuk pemanggilan fungsi push(data).

```
push(dta);
```

#### Fungsi push

```
//fungsi untuk menambahkan data pada stack
void push(float dta)
{
    if(isfull() == true) /*panggil fungsi isempty(), jika kondisi benar
pernyataan dijalankan*/
    {
        puts("Maaf, stack penuh");
    }
    else{
        stackbaru.top++;
        stackbaru.data[stackbaru.top]=dta;
    }
}
```

Pertama kali akan dicek apakah stack dalam keadaan penuh, jika true maka akan tercetak string pada layar “Maaf, stack penuh”. Jika bernilai salah maka stackbaru.top akan diincrement kemudian data yang tadi diinputkan ditambahkan pada stack.

ulang = 'y';

Setelah fungsi push() dijalankan, selanjutnya variabel ulang disetel dengan Nilai ‘y’ Sehingga setelah memilih menu push() program akan secara otomatis mengulangi prosesnya.

- e) Jika user memilih angka 3 atau *menu==’3’*, maka program akan menjalankan seluruh pernyataan yang berada di dalam kondisi ketiga.

print();

Fungsi print

```
printf("\nData yang terdapat dalam stack : \n");
printf("-----\n");
for(int i=0; i<=stackbaru.top; i++)
{
    cout<<stackbaru.data[i]<<      ";
```

Dengan memanfaatkan perulangan for, fungsi ini akan mencetak seluruh data yang berada di dalam stack.

```
cout<<"\n\nUlang ? (y/t)";
cin>>ulang;
```

Setelah data tercetak pada layar, selanjutnya akan ditampilkan string “Ulang ?(y/n)”. Jika input user adalah ‘y’ || ‘Y’ maka program akan dijalankan lagi mulai awal (diulang), tapi jika input user == ‘n’ ||‘N’ maka akan keluar dari perulangan dan selanjutnya keluar dari program.

- f) Jika user memilih angka 4 atau menu == ‘4’, maka seluruh pernyataan yang berada dalam kondisi keempat akan dijalankan.

clear();

Fungsi clear

```
//fungsi untuk membersihkan data dalam stack
void clear()
{
    stackbaru.top = -1;
    printf("\nSekarang stack kosong");
}
```

Saat fungsi ini dipanggil maka posisi stackbaru.top diinisialisasi beraada pada -1. Seperti halnya mereset ulang suatu stack yang membuat isinya akan hilang.

```
cout<<"\n\nUlang ? (y/t)";
cin>>ulang;
```

Setelah data dalam stack dibersihkan, selanjutnya akan ditampilkan string “Ulang ?(y/n)”. Jika input user adalah ‘y’ || ‘Y’ maka program akan dijalankan lagi mulai awal (diulang), tapi jika input user == ‘n’ ||’N’ maka akan keluar dari perulangan dan selanjutnya keluar dari program.

- g) Jika user memilih angka 5 atau menu==’5’ maka pernyataan yang berada pada kondisi kelima kan dijalankan

```
exit(0);
```

Pernyataan diatas digunakan untuk keluar dari program. Jadi jika kita memilih menu 5 maka kita akan keluar dari program.

### Fungsi full

```
//fungsi boolean untuk mengetahui apakah stack penuh
bool isfull()
{
    if(stackbaru.top == maxstack)
        return true;
    else
        return false;
}
```

Untuk mengetahui apakah suatu stack sedang penuh adalah dengan membandingkan stackbaru.top dengan maxstack, jika kondisi benar maka stack dalam posisi penuh, dan sebaliknya.

### Fungsi empty

```
//fungsi boolean untuk mengetahui apakah stack kosong
bool isempty()
{
    if(stackbaru.top == -1)
        return true;
    else
        return false;
}
```

Untuk mengetahui apakah suatu stack dalam keadaan kosong adalah dengan membandingkan stackbaru.top dengan -1, jika kondisi benar maka stack dalam posisi kosong, dan sebaliknya.

## BAB 7

# QUEUE

Queue merupakan jenis Linked list yang menerapkan konsep FIFO (First In First Out) atau kebalikan dari Stack (LIFO). pada Queue elemen yang dimasukkan pertama kali apabila dilakukan pemrosesan akan elemen tersebut yang akan diproses terlebih dahulu.



## Queue

Seperti halnya pada stack, sebuah queue juga dapat diimplementasikan dengan menggunakan Array, Linked list, Pointer dan Struct. Pada percobaan kali ini, akan diimplementasikan queue dengan menggunakan array satu dimensi.

Terdapat 2 Operasi dasar dari Queue:

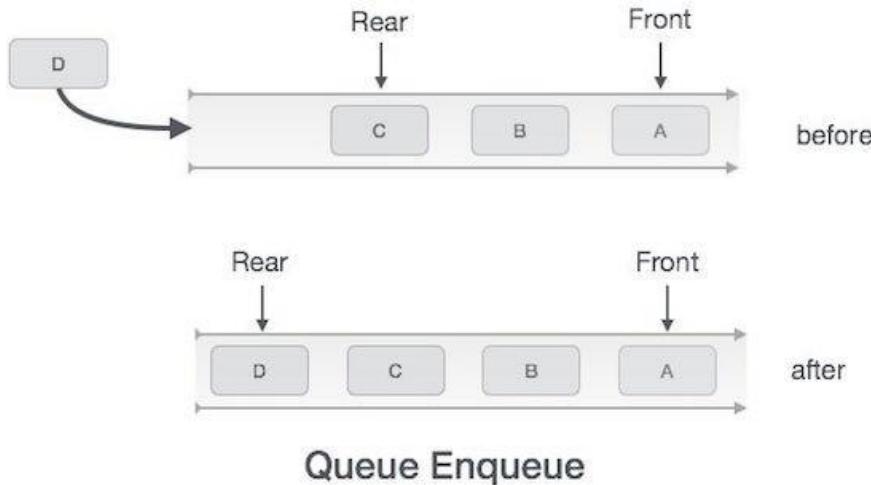
- enqueue() – Proses penambahan elemen di posisi belakang
- dequeue() – Proses pengambilan elemen di posisi depan

Untuk membuat sebuah operasi queue menjadi efisien, dibutuhkan beberapa fungsi antara lain:

- peek() – Mengambil elemen yang berada di posisi depan tanpa harus menghapusnya.
- isfull() – Mengecek apakah queue sudah penuh atau tidak
- isempty() – Mengecek apakah queue dalam keadaan kosong.

Langkah-langkah yang dilakukan untuk menambahkan data ke dalam antrian (enqueue) adalah sebagai berikut.

- **Step 1** – Cek apakah queue dalam kondisi penuh
- **Step 2** – Jika queue dalam keadaan penuh, hasilkan overflow error dan keluar.
- **Step 3** – Jika queue tidak penuh, pindahkan pointer rear menuju ke ruang kosong.
- **Step 4** – Tambahkan elemen data ke lokasi queue yang ditunjukan oleh pointer rear.
- **Step 5** – Proses berhasil.



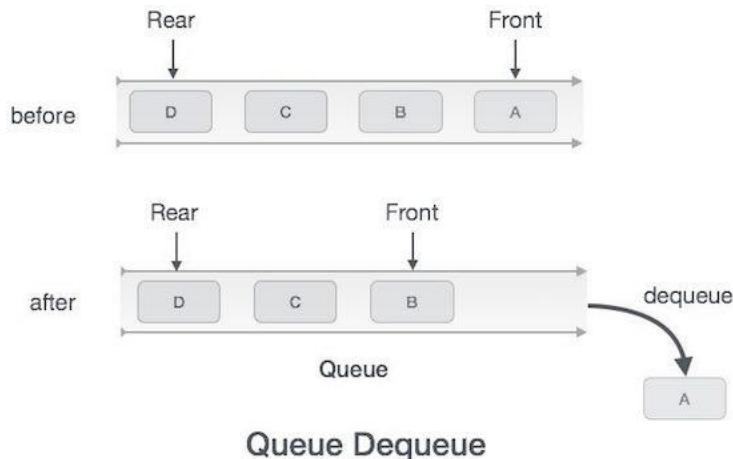
Algoritma dari proses Enqueue adalah sebagai berikut:

```

1 void insert(int data) {
2
3     if(!isFull()) {
4
5         if(rear == MAX-1) {
6             rear = -1;
7         }
8
9         intArray[++rear] = data;
10        itemCount++;
11    }
12 }
```

Mengakses data dari queue terdiri dari dua proses, mengakses data yang ditunjukan pointer front dan menghapus data setelah diakses. Langkah-langkah melakukan proses dequeue adalah sebagai berikut.

- **Step 1** – Cek apakah queue dalam keadaan kosong atau tidak.
- **Step 2** – Jika queue dalam keadaan kosong, hasilkan overflow error dan keluar.
- **Step 3** – Jika queue tidak kosong, akses data yang ditunjukan oleh pointer front.
- **Step 4** – Tingkatkan pointer front menuju ke element data berikutnya.
- **Step 5** – Return success.



Algoritma dari proses Dequeue adalah sebagai berikut:

```

1 int removeData() {
2     int data = intArray[front++];
3
4     if(front == MAX) {
5         front = 0;
6     }
7
8     itemCount--;
9     return data;
10}

```

Source code dari proses queue adalah sebagai berikut:

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <stdbool.h>
5
6 #define MAX 6
7
8 int intArray[MAX];
9 int front = 0;
10 int rear = -1;
11 int itemCount = 0;
12
13 int peek() {
14     return intArray[front];
15 }
16
17 bool isEmpty() {
18     return itemCount == 0;
19 }

```

```
20 bool isFull() {
21     return itemCount == MAX;
22 }
23
24
25 int size() {
26     return itemCount;
27 }
28
29 void insert(int data) {
30
31     if(!isFull()) {
32
33         if(rear == MAX-1) {
34             rear = -1;
35         }
36
37         intArray[++rear] = data;
38         itemCount++;
39     }
40 }
41
42 int removeData() {
43     int data = intArray[front++];
44
45     if(front == MAX) {
46         front = 0;
47     }
48
49     itemCount--;
50     return data;
51 }
52
53 int main() {
54     /* insert 5 items */
55     insert(3);
56     insert(5);
57     insert(9);
58     insert(1);
59     insert(12);
60
61     // front : 0
62     // rear : 4
63     // -----
64     // index : 0 1 2 3 4
65     // -----
66     // queue : 3 5 9 1 12
67     insert(15);
68
69     // front : 0
```

```

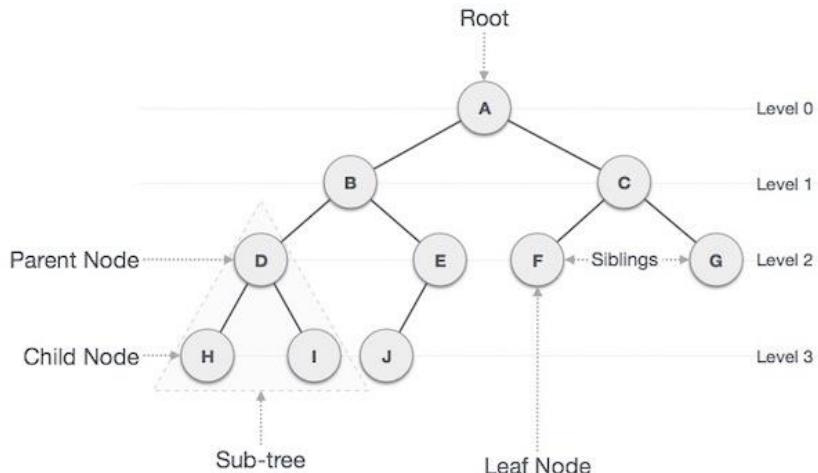
70 // rear : 5
71 // -----
72 // index : 0 1 2 3 4 5
73 // -----
74 // queue : 3 5 9 1 12 15
75
76 if(isFull()){
77     printf("Queue is full!\n");
78 }
79
80 // remove one item
81 int num = removeData();
82
83 printf("Element removed: %d\n",num);
84 // front : 1
85 // rear : 5
86 // -----
87 // index : 1 2 3 4 5
88 // -----
89 // queue : 5 9 1 12 15
90 // insert more items
91 insert(16);
92 // front : 1
93 // rear : -1
94 // -----
95 // index : 0 1 2 3 4 5
96 // -----
97 // queue : 16 5 9 1 12 15
98
99 // As queue is full, elements will not be inserted.
100 insert(17);
101 insert(18);
102
103 // -----
104 // index : 0 1 2 3 4 5
105 // -----
106 // queue : 16 5 9 1 12 15
107 printf("Element at front: %d\n",peek());
108
109 printf("-----\n");
110 printf("index : 5 4 3 2 1 0\n");
111 printf("-----\n");
112 printf("Queue: ");
113
114 while(!isEmpty()) {
115     int n = removeData();
116     printf("%d ",n);
117 }
118 }
119

```

## BAB 8

### TREE

Kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layaknya struktur sebuah pohon. Struktur pohon adalah suatu cara merepresentasikan suatu struktur hirarki (one-to-many) secara grafis yang mirip sebuah pohon, walaupun pohon tersebut hanya tampak sebagai kumpulan node-node dari atas ke bawah. Suatu struktur data yang tidak linier yang menggambarkan hubungan yang hirarkis (one-to-many) dan tidak linier antara elemen-elemennya.



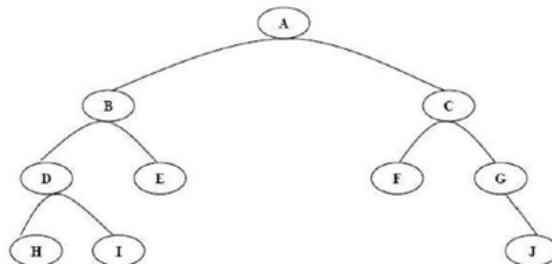
Istilah-istilah penting pada sebuah tree.

|                   |  |
|-------------------|--|
| <b>Predecesor</b> | Node yang berada diatas node tertentu.   |
| <b>Successor</b>  | Node yang berada dibawah node tertentu.  |
| <b>Ancestor</b>   | Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama |
| <b>Descendant</b> | Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama |
| <b>Parent</b>     | Predecessor satu level di atas suatu node.   |
| <b>Child</b>      | Successor satu level di bawah suatu node.  |
| <b>Sibling</b>    | Node-node yang memiliki parent yang sama   |
| <b>Subtree</b>    | Suatu node beserta descendantnya.  |
| <b>Size</b>       | Banyaknya node dalam suatu tree  |
| <b>Height</b>     | Banyaknya tingkatan dalam suatu tree   |
| <b>Root</b>       | Node khusus yang tidak memiliki predecessor.                                       |
| <b>Leaf</b>       | Node-node dalam tree yang tidak memiliki successor.                                |
| <b>Degree</b>     | Banyaknya child dalam suatu node   |

Jenis-Jenis Tree

## Binary Tree

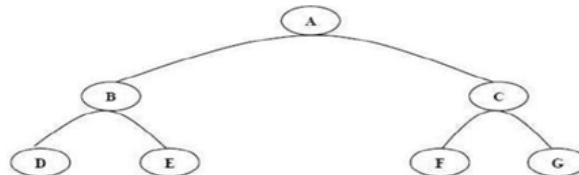
Tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua sub pohon dan kedua subpohon harus terpisah.



Binary Tree

## Full Binary Tree

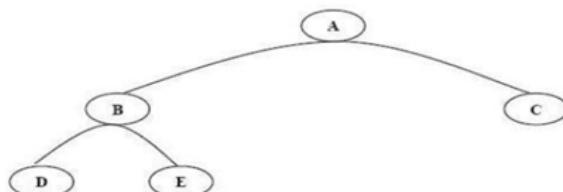
Semua node, kecuali leaf pasti memiliki 2 anak dan tiap subpohon memiliki panjang path yang sama.



Full Binary Tree

## Complete Binary Tree

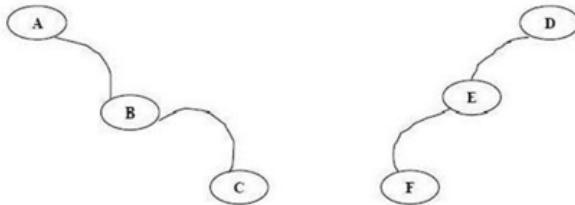
Tree yang mirip dengan full binary tree, tapi tiap subtree boleh memiliki panjang path yang berbeda dan tiap node (kecuali leaf) memiliki 2 anak.



Complete Binary Tree

## Skewed Binary Tree

Binary tree yang semua nodenya (kecuali leaf) hanya memiliki satu anak.



Skewed Binary Tree

```

1 //header file
2 #include <stdio.h>
3 // #include <conio.h>
4 #include <stdlib.h>
5
6 //pendeklarasian struct sebuah tree awal
7 struct Node{
8     int data;
9     Node *kiri;
10    Node *kanan;
11};
12
13 //fungsi untuk menambahkan node baru
14 void tambah(Node **root, int databaru)
15 {
16     //jika root masih kosong
17     if((*root) == NULL)
18     {
19         //pembuatan node baru
20         Node *baru;
21         //pengalokasian memori dari node yang telah dibuat
22         baru = new Node;
23         //inisialisasi awal node yang baru dibuat
24         baru->data = databaru;
25         baru->kiri = NULL;
26         baru->kanan = NULL;
27         (*root) = baru;
28         (*root)->kiri = NULL;
29         (*root)->kanan = NULL;
30         printf("Data bertambah!\n");
31     }
32     //jika data yang akan dimasukkan lebih kecil daripada elemen root, maka akan diletakkan di node
33     sebelah kiri.
34     else if(databaru<(*root)->data)
35         tambah(&(*root)->kiri, databaru);
36     //jika data yang akan dimasukkan lebih besar daripada elemen root, maka akan diletakkan di node
37     sebelah kanan

```

```

38     else if(databaru>(*root)->data)
39         tambah(&(*root)->kanan, databaru);
40 //jika saat dicek data yang akan dimasukkan memiliki nilai yang sama dengan data pada root
41     else if(databaru == (*root)->data)
42         printf("Data sudah ada!\n");
43 }
44
45 //fungsi yang digunakan untuk mencetak tree secara preOrder
46 void preOrder(Node *root)
47 {
48     if(root != NULL){
49         printf("%d ", root->data);
50         preOrder(root->kiri);
51         preOrder(root->kanan);
52     }
53 }
54
55 //fungsi yang digunakan untuk mencetak tree secara inOrder
56 void inOrder(Node *root)
57 {
58     if(root != NULL){
59         inOrder(root->kiri);
60         printf("%d ", root->data);
61         inOrder(root->kanan);
62     }
63 }
64
65 //fungsi yang digunakan untuk mencetak tree secara postOrder
66 void postOrder(Node *root)
67 {
68     if(root != NULL){
69         postOrder(root->kiri);
70         postOrder(root->kanan);
71         printf("%d ", root->data);
72     }
73 }
74
75 //fungsi utama
76 int main()
77 {
78     //deklarasikan variabel
79     int pil, data;// c;
80     Node *pohon; /*t;
81     pohon = NULL; //inisialisasi node pohon
82     //perulangan do-while
83     do
84     {
85 //        system("cls"); //bersihkan layar
86         printf("\t#PROGRAM TREE C++#");
87         printf("\n\t=====");

```

```

88    printf("\nMENU");
89    printf("\n---\n");
90    printf("1. Tambah\n");
91    printf("2. Lihat pre-order\n");
92    printf("3. Lihat in-order\n");
93    printf("4. Lihat post-order\n");
94    printf("5. Exit\n");
95    printf("Pilihan : ");
96    scanf("%d", &pil);
97    switch(pil)
98    {
99        //jika pil bernilai 1
100       case 1 :
101           printf("\nINPUT : ");
102           printf("\n-----");
103           printf("\nData baru : ");
104           scanf("%d", &data);
105           //panggil fungsi untuk menambah node yang berisi data pada tree
106           tambah(&pohon, data);
107           break;
108
109        //jika pil bernilai 2
110       case 2 :
111           printf("\nOUTPUT PRE ORDER : ");
112           printf("\n-----\n");
113           if(pohon!=NULL)
114               //panggil fungsi untuk mencetak data secara preOrder
115               preOrder(pohon);
116           else
117               printf("Masih kosong!");
118           break;
119
120        //jika pil bernilai 3
121       case 3 :
122           printf("\nOUTPUT IN ORDER : ");
123           printf("\n-----\n");
124           if(pohon!=NULL)
125               //panggil fungsi untuk mencetak data secara inOrder
126               inOrder(pohon);
127           else
128               printf("Masih kosong!");
129           break;
130
131        //jika pil bernilai 4
132       case 4 :
133           printf("\nOUTPUT POST ORDER : ");
134           printf("\n-----\n");
135           if(pohon!=NULL)
136               //panggil fungsi untuk mencetak data secara postOrder
137               postOrder(pohon);

```

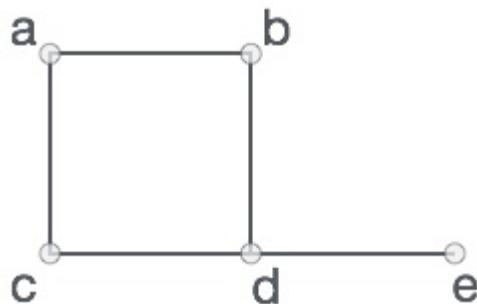
```
138     else
139         printf("Masih kosong!");
140     break;
141 }
142 //_getch();
143 }while(pil != 5); //akan diulang jika input tidak samadengan 5
return EXIT_FAILURE;
}
```

## BAB 9

### GRAF

Graph adalah representasi bergambar dari sekumpulan objek di mana beberapa pasang benda dihubungkan oleh tautan. Objek yang saling berhubungan diwakili oleh titik yang disebut sebagai simpul (**vertices**), dan tautan yang menghubungkan simpul disebut tepi (**edges**).

Secara formal, grafik adalah sepasang himpunan (**V, E**), di mana **V** adalah himpunan simpul dan **E** adalah himpunan tepi, yang menghubungkan pasangan simpul. Perhatikan graph berikut.



Pada graph di atas, terlihat himpunan untuk masing-masing **V** dan **E** yaitu:

$$V = \{a, b, c, d, e\}$$

$$E = \{ab, ac, bd, cd, de\}$$

#### Graph Data Structure

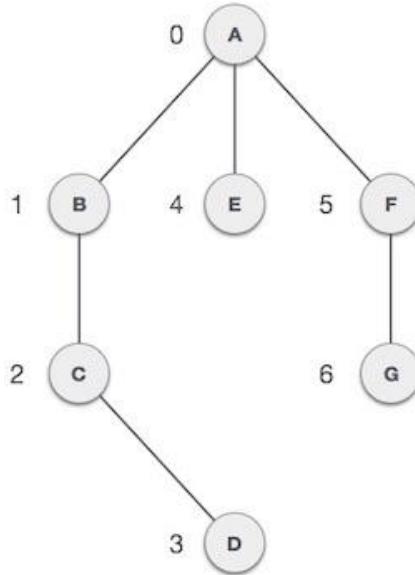
Representasi matematika untuk graf dapat direpresentasikan dalam struktur data. Kita dapat merepresentasikan grafik menggunakan larik simpul dan larik dua dimensi tepi. Sebelum melangkah lebih jauh, mari kita membiasakan diri dengan beberapa istilah penting.

**Vertex** – Setiap node grafik direpresentasikan sebagai titik. Dalam contoh berikut, lingkaran berlabel mewakili simpul. Jadi, A ke G adalah simpul. Kami dapat mewakilinya menggunakan larik seperti yang ditunjukkan pada gambar berikut. Di sini A dapat diidentifikasi dengan indeks 0. B dapat diidentifikasi menggunakan indeks 1 dan seterusnya.

**Edge** – Setiap edge mewakili jalur antara dua simpul atau garis antara dua simpul. Dalam contoh berikut, garis-garis dari A ke B, B ke C, dan sebagainya mewakili sisi-sisi. Kita dapat menggunakan larik dua dimensi untuk mewakili larik seperti yang ditunjukkan pada gambar berikut. Di sini AB dapat direpresentasikan sebagai 1 pada baris 0, kolom 1, BC sebagai 2 pada baris 1, kolom 2 dan seterusnya, menjaga kombinasi lain sebagai 0.

**Adjacency** – Dua simpul atau simpul berdekatan jika mereka terhubung satu sama lain melalui suatu tepi. Dalam contoh berikut, B berbatasan dengan A, C bersebelahan dengan B, dan seterusnya.

- **Path** – Path merepresentasikan urutan tepi antara dua simpul. Dalam contoh berikut, ABCD mewakili jalur dari A ke D.



## Basic Operations

Berikut ini adalah operasi utama dasar Grafik.

- **Add Vertex** – Menambahkan titik ke grafik..
- **Add Edge** – Menambahkan tepi antara dua simpul grafik.
- **Display Vertex** – Menampilkan vertex dari graph.

|  |   |
|--|---|
|  | <pre> 1 #include&lt;iostream&gt; 2 #include&lt;cstdlib&gt; 3 4 using namespace std; 5 6 int main(){ 7     bool ketemu,nolsemua; 8     int matrix[10] [10]; 9     int i,j,jumlah_simpul,jumlah_sisi,asal,tujuan; 10 11    //inisialisasi matrix 12    cout&lt;&lt;"jumlah simpul:"; 13    cin&gt;&gt;jumlah_simpul; 14    cout&lt;&lt;"jumlah_sisi:"; 15    cin&gt;&gt;jumlah_sisi;   </pre> |
|--|---|

```

16 for (i=1;i<=jumlah_simpul;i++)
17     for (j=1;j<=jumlah_simpul;j++)
18         matrix[i][j]=0;
19
20 //isi matrix sesuai input graf
21 for (i=1;i<=jumlah_sisi;i++)
22 {
23     cout<<"simpul asal:";
24     cin>>asal;
25     cout<<"simpul tujuan:";
26     cin>>tujuan;
27     matrix[asal][tujuan]=1;
28     matrix[tujuan][asal]=1;
29 }
30
31 //telusuri graf
32 i=1;nolsemua=false;
33 while (i<=jumlah_simpul && !nolsemua)
34 {
35     j=1;ketemu=false;
36     while (j<=jumlah_simpul && !ketemu)
37     {
38         if (matrix[i][j]==1)
39             ketemu=true;
40         else
41             j++;
42     }
43     if (!ketemu)
44         nolsemua=true;
45     else
46         i++;
47 }
48
49 if(nolsemua)
50     cout<<"graf tidak terhubung"<<endl;
51 else
52     cout<<"graf terhubung"<<endl;
53
54 //system("PAUSE");
55 return EXIT_SUCCESS;
56 }
```