STRUKTUR DATA

Modul Praktikum

JULY 5, 2018

PROGRAM STUDI TEKNIK INFORMATIKA
Jurusan Teknik Elektro Fakultas Teknik Universitas Lampung

DAFTAR ISI

KATA PENGANTAR

Assalamualaikum Wr. Wb.

Alhamdulillah, kami ucapkan syukur ke hadirat Allah SWT, bahwa berkat rahmat dan karunia-Nya, kami masih diberikan kesempatan untuk melaksanakan kegiatan Praktikum Struktur Data. Praktikum Struktur Data adalah implementasi atau kegiatan praktik untuk menerapkan teori yang sudah dipelajari dalam mata kuliah Struktur Data. Ilmu yang dipelajari dalam praktikum ini membangun pondasi yang kuat dalam menggunakan struktur data untuk pembangunan perangkat lunak pada komputer.

Data merupakan entitas yang dapat menjadi kompleks. Dengan memahami bagaimana mengelola data yang kompleks, pembuatan perangkat lunak dapat mengolah data menjadi informasi yang dapat berguna. Sebagai insan yang berkecimpung di bidang teknologi informasi, keahlian untuk dapat mengelola data berdasarkan aturan-aturan yang berlaku pada komputer menjadi hal yang krusial untuk dikuasai.

Ucapan terima kasih, kami sampaikan kepada Ketua Program Studi Teknik Informatika, Ketua Jurusan Teknik Elektro Unila, Asisten Praktikum Struktur Data, segenap dosen, teknisi, karyawan, dan seluruh mahasiswa Jurusan Teknik Elektro Universitas Lampung, yang dalam hal ini termasuk S1 Teknik Informatika, S1 Teknik Elektro, dan S2 Teknik Elektro, atas dukungan dan perhatian yang telah diberikan.

Panduan Praktikum ini selalu membutuhkan peningkatan, untuk itu kritik dan saran yang bersifat membangun untuk Panduan Praktikum ini akan kami perhatikan dan tindak lanjuti dengan seksama. Semoga ilmu yang diperoleh dalam Praktikum ini dapat dimanfaatkan dengan sebaik-baiknya.

Wassalamualaikum wr. wb.

Bandar Lampung, 1 Agustus 2018
Dosen Penanggung Jawab
Praktikum Struktur Data

Meizano A.M. Djausal, M.T.
NIP 19810528 201212 1 001

TATA TERTIB PRAKTIKUM

1. Mahasiswa yang diizinkan mengikuti praktikum adalah yang telah terdaftar dan memenuhi syarat yang telah ditentukan.
2. Praktikan dilarang mengoperasikan alat tanpa seizin asisten.
3. Praktikum dilaksanakan sesuai jadwal dan praktikan harus hadir 15 menit sebelum praktikum dimulai.
4. Praktikan harus berpakaian rapi dan tidak diperkenankan memakai kaos oblong.
5. Praktikan dilarang membuat kegaduhan selama berada dalam laboratorium dan wajib menjaga kebersihan di dalam maupun di halaman laboratorium.
6. Praktikan wajib mengerjakan tugas pendahuluan dari setiap percobaan sebelum mengikuti praktikum.
7. Ketidakhadiran peserta dalam suatu praktikum harus atas sepengetahuan asisten yang bersangkutan. Ketidakhadiran tanpa izin asisten akan mengurangi nilai laporan dari percobaan sebesar 20%.
8. Praktikan harus melaksanakan asistensi kepada asisten yang bersangkutan selama penulisan laporan.
9. Pelanggaran terhadap tata tertib akan diberikan sanksi: "TIDAK DIPERKENANKAN MENGIKUTI PRAKTIKUM"

PENGETAHUAN WAJIB

1. Memahami cara kerja dan mampu mengoperasikan komputer.
2. Mengetahui konsep file (berkas), dapat melakukan operasi dasar file (copy, paste, delete, dsb).
3. Dapat memanipulasi file menggunakan text editor.
4. Memahami konsep dasar pemrograman komputer.
5. Mampu menggunakan CLI (Command Line Interface).
6. Dapat membedakan antara huruf besar, huruf kecil, koma (,), titik koma (;), titik dua (:), kutip tunggal (‘), kutip ganda (“), dan lainnya yang diperlukan.

1 PERCOBAAN 1: ARRAY

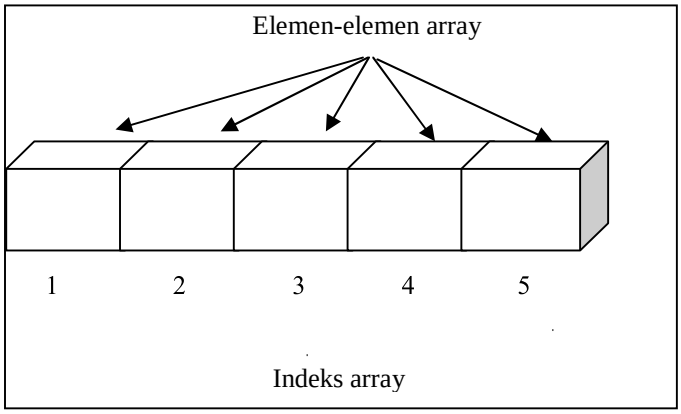
1.1 TUJUAN PERCOBAAN

- Dapat melakukan pembuatan array
- Dapat memahami penggunaan array

1.2 TINJAUAN PUSTAKA

1.2.1 Pengertian Array

Variabel pada program biasanya hanya dapat menampung sebuah nilai numerik atau string pada suatu waktu. Apabila ingin memberikan nilai yang baru pada variabel tersebut, maka nilai lama akan hilang karena digantikan oleh nilai yang baru. Bagaimana apabila beberapa nilai/data ingin disimpan dalam sebuah variabel dengan nama yang sama tetapi semua nilai tetap tersimpan? Solusi yang dapat dilakukan adalah dengan menggunakan indeks pada nama variabel tersebut. Cara ini disebut dengan array.



Gambar 1-1. Indeks Array

Array adalah struktur data yang menyimpan sekumpulan elemen yang bertipe sama, dan setiap elemen diakses langsung melalui indeksnya. Indeks array haruslah tipe data yang menyatakan keterurutan, misalnya: integer atau string. Array dapat dianalogikan sebagai sebuah lemari yang memiliki sederetan kotak penyimpanan yang diberi nomor berurutan. Untuk menyimpan atau mengambil sesuatu dari kotak tertentu, kita hanya cukup mengetahui nomor kotaknya saja.

Array adalah suatu tipe data terstruktur yang terdapat dalam memori yang terdiri dari sejumlah elemen (tempat) yang mempunyai tipe data yang sama dan merupakan gabungan dari beberapa variabel sejenis serta memiliki jumlah komponen yang jumlahnya tetap.

Array (biasa juga disebut larik) merupakan tipe data terstruktur yang berguna untuk menyimpan sejumlah data yang bersifat sama. Bagian yang menyusun array biasa dinamakan elemen array. Masing-masing elemen dapat diakses tersendiri, melalui indeks array.

Elemen-elemen dari array tersusun secara sequential dalam memori komputer. Array dapat berupa satu dimensi, dua dimensi, tiga dimensi ataupun banyak dimensi.

1.2.2 Jenis-Jenis Array

a. Array satu dimensi

Array satu dimensi dapat dikatakan sebagai suatu daftar linier atau sebuah kolom. Array berdimensi satu dapat digambarkan sebagai kotak panjang yang terdiri atas beberapa kotak kecil.

Bentuk umum dari array jenis ini adalah:

```
tipe_data nama_variabel_array[besar_dimensi];
```

Contoh:

```
int x[10];
```

$$x = \begin{bmatrix} x[1] \\ x[2] \\ ..... \\ ..... \\ x[10] \end{bmatrix}$$

Pada contoh ini, X dapat menampung 10 buah elemen bertipe Real. Yang menjadi kata-kata kunci pendeklarasian array adalah kata cadang ARRAY. Banyaknya komponen dalam suatu larik ditunjukkan oleh suatu indeks yang disebut dengan tipe indeks (index type). Tiap-tiap komponen larik dapat diakses dengan menunjukkan nilai indeksnya.

b. Array Multidimensi

Contoh umum untuk array jenis ini adalah array dua dimensi dan array tiga dimensi. Array dua dimensi ini dapat dianggap sebagai sebuah matriks yang jumlah kolomnya lebih dari satu. Sedangkan, array tiga dimensi bisa digunakan untuk pentabelan data (baris, kolom, isi data).

Bentuk umum array 2 dimensi:

```
tipe_data nama_array[besar_dimensi1][besar_dimensi2];
```

Bentuk deklarasi array 3 dimensi:

```
tipe_data nama_array[besar_dimensi1][besar_dimensi2]
[besar_dimensi3];
```

Contoh:

```
int y[10][10];

int z[10][10][10];
```

1.3 PERCOBAAN

1.3.1 Percobaan I-1: Array 1 Dimensi

Ketik kode program di bawah ini:

```cpp
#include <iostream>
using namespace std;

int main()
{
    int a[8];
    cout << &a << endl;
    for (int i=0; i<8; i++) {
        cout << &a[i] << endl;
    }
    return 0;
}
```

Kode 1-1. ArrayIntAddress.cpp

```cpp
#include <iostream>
using namespace std;

int main()
{
    char a[8];
    cout << &a << endl;
    for (int i=0; i<8; i++) {
        cout << &a[i] << endl;
    }
    return 0;
}
```

Kode 1-2. ArrayCharAddress.cpp

```cpp
#include <iostream>
using namespace std;

int main()
{
    int a[8];

    for (int i=0; i<8; i++) {
        cin >> a[i];
    }

    for (int i=0; i<8; i++) {
        cout << a[i] << endl;
    }
```

```
    return 0;
}
```

Kode 1-3. ArrayInputOutput.cpp

```
#include <iostream>
using namespace std;

int main()
{
    char a[15] = "informatika";
    cout << a << endl;
    a[9] = 'c';
    a[10] = 's';
    cout << a << endl;
    cin >> a;
    cout << a << endl;

    return 0;
}
```

Kode 1-4. ArrayCharacter.cpp

- Jelaskan kode program di atas!

1.3.2 Percobaan I-2: Array 2 Dimensi

Ketik kode program di bawah ini:

```
#include <iostream>
using namespace std;

int main()
{
    char a[3][5];
    for (int i=0; i<3; i++) {
        cout << &a[i] << endl;
        for (int j=0; j<5; j++) {
            cout << &a[i][j] << endl;
        }
    }
        cout << &a << endl;

    return 0;
}
```

Kode 1-5. Array2DCharAddress.cpp

```
#include <iostream>
using namespace std;

int main()
{
```

```
    int a[3][5];
    for (int i=0; i<3; i++) {
        cout << &a[i] << endl;
        for (int j=0; j<5; j++) {
            cout << &a[i][j] << endl;
        }
    }
        cout << &a << endl;

    return 0;
}
```

Kode 1-6. Array2DIntAddress.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int a[2][3];
    for (int i=0; i<2; i++) {
        for (int j=0; j<3; j++) {
            cin >> a[i][j];
        }
    }

    for (int i=0; i<2; i++) {
        for (int j=0; j<3; j++) {
            cout << a[i][j] << endl;
        }
    }

    return 0;
}
```

Kode 1-7. Array2DInputOutput.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int a[2][3];
    for (int i=0; i<2; i++) {
        for (int j=0; j<3; j++) {
            cin >> a[i][j];
        }
    }
```

```
    a[0][1] = a[1][2] + a[0][0];
    a[1][1] = a[0][0] * 20;

    for (int i=0; i<2; i++) {
        for (int j=0; j<3; j++) {
            cout << a[i][j] << endl;
        }
    }

    return 0;
}
```

Kode 1-8. Array2DIndex.cpp

- Jelaskan kode program di atas!

1.4 TUGAS AKHIR

1. Buatlah program untuk penjumlahan matriks menggunakan array 2 dimensi. Jelaskan hasil dari eksekusi program!

2 PERCOBAAN 2: SORTING

2.1 TUJUAN PERCOBAAN

- Memahami tujuan dari sorting
- Dapat menggunakan sorting

2.2 TINJAUAN PUSTAKA

Data terkadang berada dalam bentuk yang tidak berpola ataupun dengan pola tertentu yang tidak diinginkan, namun dalam penggunaanya, ada kecenderungan untuk menggunakan data-data tersebut dalam bentuk yang rapi atau berpola sesuai dengan yang kita inginkan. Oleh karena itu, proses sorting adalah proses yang sangat penting dalam struktur data, terlebih untuk pengurutan data yang bertipe numerik ataupun leksikografi (urutan secara abjad sesuai kamus).

Sorting adalah proses menyusun kembali data yang sebelumnya telah disusun dengan suatu pola tertentu ataupun secara acak, sehingga menjadi tersusun secara teratur menurut aturan tertentu. Pada umumnya ada 2 macam pengurutan, yaitu: pengurutan secara ascending (urut naik) dan pengurutan secara descending (urut turun).

Banyak klasifikasi yang dapat digunakan untuk mengklasifikasikan algoritma-algoritma pengurutan, misalnya secara kompleksitas, teknik yang dilakukan, stabilitas, memori yang digunakan, rekursif atau tidak ataupun proses yang terjadi. Secara umum, metode pengurutan dapat dikelompokkan dalam 2 kategori, yaitu:

1. Metode pengurutan sederhana (elementary sorting methods). Metode pengurutan sederhana meliputi bubble sort, selection sort, dan insertion sort.
2. Pengurutan lanjut (advanced sorting methods). Metode pengurutan lanjut diantaranya shell sort, quick sort, merge sort dan radix sort.

Algoritma-algoritma ini tentu saja akan mempunyai efek yang berbeda dalam setiap prosesnya, ada yang mudah digunakan, ada yang mempunyai proses yang sangat cepat. Pemilihan algoritma untuk sorting ini tidak hanya asal saja dipilih. Pemilihan ini semestinya berdasarkan kebutuhan yang diperlukan. Tidak semua algortima yang pendek itu buruk dan tidak semua algoritma yang super cepat juga akan baik dalam semua kondisi. Misal: algoritma quick sort adalah algoritma sorting yang tercepat dalam proses pencariannya, namun jika data yang akan diurutkan ternyata sudah hampir terurut atau tidak terlalu banyak, maka algoritma ini malah akan memperlama proses pengurutan itu sendiri, karena akan banyak perulangan tidak perlu yang dilakukan dalam proses sorting ini.

2.2.1 Bubble Sort

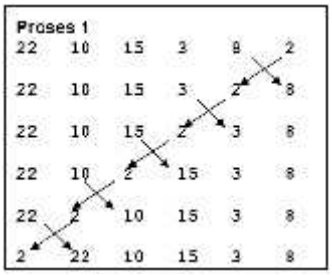
Keuntungan dari algoritma sorting ini adalah karena paling mudah, dan dapat dijalankan dengan cukup cepat dan efisien untuk mengurutkan list yang urutannya sudah hampir benar. Namun algoritma ini paling lambat dan termasuk sangat tidak efisien untuk dilakukan dibandingkan dengan algoritma yang lain apalagi pengurutan dilakukan terhadap elemen yang banyak jumlahnya. Untuk itu biasanya *bubble sort* hanya digunakan

untuk mengenalkan konsep dari algoritma sorting pada pendidikan komputer karena idenya yang cukup sederhana, yaitu mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya. Kompleksitas untuk algoritma ini adalah $O(n^2)$.

(1) Konsep algoritma *bubble sort*

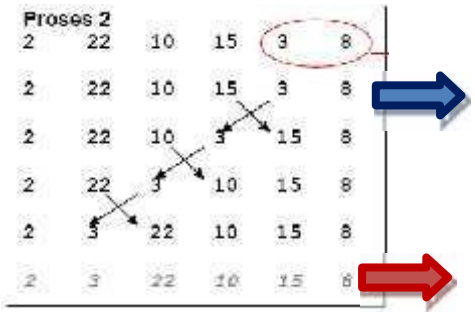
a. Algoritma dimulai dari elemen paling awal.
b. buah elemen pertama dari list dibandingkan. Jika elemen pertama lebih besar dari elemen kedua atau sebaliknya (urut secara *ascending* atau *descending*),dilakukan pertukaran.
c. Langkah 2 dan 3 dilakukan lagi terhadap elemen kedua dan ketiga, seterusnya 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,...,n-1 harus memenuhi keterurutan, yaitu untuk pengurutan:
    - Ascending : x[j] < x[j+1]
    - Descending : x[j] > x[j+1]

(2) Implementasi *bubble sort*

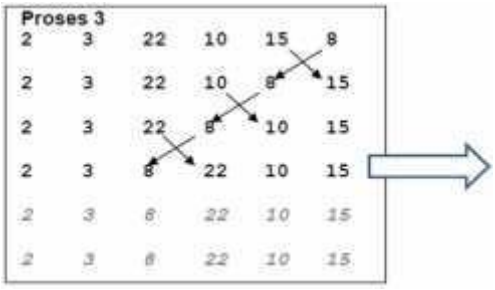


Gambar 2-1. Ilustrasi algoritma bubble sort untuk pengurutan secara ascending

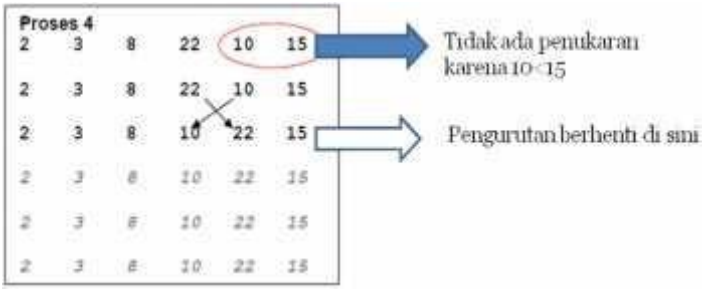
a. Pada Gambar 1, pegecekan dimulai dari data yang paling akhir, kemudian di bandingkan dengan data di depannya, jika data di depannya lebih besar maka akan ditukar
b. Pada proses kedua, proses pertama diulangi dan pengecekan dilakukan sampai dengan data ke-2 karena data pertama pasti sudah paling kecil.
c. 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, dengan n adalah jumah data yang akan diurutkan.



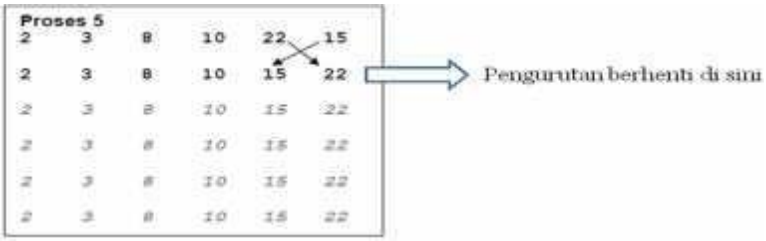
Tidak ada penukaran karena 3<8



Pengurutan Berhenti Di Sini



Pengurutan Berhenti Di Sini



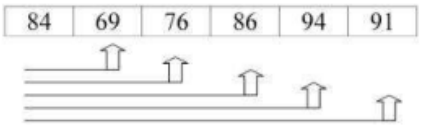
Gambar 2-2. Ilustrasi algoritma bubble sort pada proses 3 s.d 5

2.2.2 Exchange Sort

Exchange Sort sangat mirip dengan Bubble Sort. Pebedaannya adalah dalam hal bagaimana membandingkan antar elemen-elemennya.

Exchange sort membandingkan suatu elemen dengan elemen-elemen lainnya dalam array tersebut, dan melakukan pertukaran elemen jika perlu. Jadi ada elemen yang selalu menjadi elemen pusat (pivot). Sedangkan Bubble sort akan membandingkan elemen pertama/terakhir dengan elemen sebelumnya/sesudahnya, kemudian elemen

sebelum/sesudahnya itu akan menjadi pusat (pivot) untuk dibandingkan dengan elemen sebelumnya/sesudahnya lagi, begitu seterusnya.



Proses 1

Pivot (Pusat)

84	69	76	86	94	91
84	69	76	86	94	91
84	69	76	**86**	94	91
86	69	76	**84**	**94**	91
94	69	76	84	**86**	91
94	69	76	84	86	91

Proses 2

Pivot (Pusat)

94	**69**	**76**	84	86	91
94	**76**	**69**	**84**	86	91
94	**84**	69	**76**	**86**	91
94	**86**	69	76	**84**	**91**
94	**91**	69	76	84	86

Proses 3

Pivot (Pusat)

94	91	**69**	**76**	84	86
94	91	**76**	**69**	**84**	86
94	91	**84**	69	**76**	**86**
94	91	**86**	69	76	84

Proses 4

Pivot (Pusat)

94	91	86	**69**	**76**	84
94	91	86	**76**	**69**	**84**
94	91	86	**84**	69	**76**

Proses 5

Pivot (Pusat)

94	91	86	84	**69**	**76**
94	91	86	84	**76**	**69**

Gambar 2-3. Ilustrasi algoritma exchange sort

2.2.3 Selection Sort

Prinsip utama algoritma dalam klasifikasi ini adalah mencari elemen yang tepat untuk diletakkan di posisi yang telah diketahui, dan meletakkannya di posisi tersebut setelah data tersebut ditemukan. Algoritma yang dapat diklasifikasikan ke dalam kategori ini adalah : *Selection sort, dan Heapsort.*

Selection Sort merupakan kombinasi antara *sorting* dan *searching*. Untuk setiap proses, akan dicari elemen-elemen yang belum diurutkan yang memiliki nilai terkecil atau terbesar akan dipertukarkan ke posisi yang tepat di dalam array.

Kelebihan dan kekurangan Selection Sort:

1. Kompleksitas selection sort relatif lebih kecil
2. Mudah menggabungkannya kembali, tetapi sulit membagi masalah
3. Membutuhkan meteode tambahan

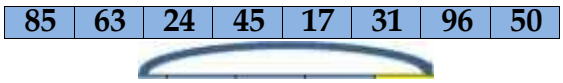
(1) Konsep algoritma *selection sort*

Untuk putaran pertama, akan dicari data dengan nilai terkecil dan data ini akan ditempatkan di indeks terkecil (data[1]), pada putaran kedua akan dicari data kedua terkecil, dan akan ditempatkan di indeks kedua (data[2]). Selama proses, pembandingan dan pengubahan hanya dilakukan pada indeks pembanding saja, pertukaran data secara fisik terjadi pada akhir proses.

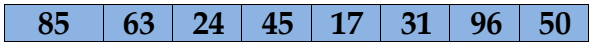
Teknik pengurutan dgn cara pemilihan elemen atau proses kerja dgn memilih elemen data terkecil untuk kemudian dibandingkan & ditukarkan dgn elemen pada data awal, dan seterusnya sampai dengan seluruh elemen sehingga akan menghasilkan pola data yg telah disort.

(2)* Implementasi algoritma *selection sort

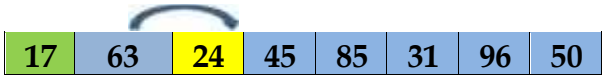
- Bila diketahui data sebelum dilakukan proses sortir

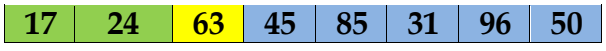


- *Iterasi 1:*



- **Iterasi 2:**





- **Iterasi 3:**

17	24	63	45	85	31	96	50

17	24	31	45	85	63	96	50

- **Iterasi 4:**
  Pada iterasi ke empat karena data terkecil adalah data ke-4, maka tidak dilakukan proses pertukaran

17	24	31	45	85	63	96	50

- **Iterasi 5:**

17	24	31	45	85	63	96	50

17	24	31	45	50	63	96	85

- **Iterasi 6:**
  Pada iterasi ke enam karena data terkecil adalah data ke-6, maka tidak dilakukan proses pertukaran seperti pada iterasi ke-4

17	24	31	45	50	63	96	85

- **Iterasi 7:**

17	24	31	45	50	63	96	85

17	24	31	45	50	63	85	96

2.2.4 Insertion Sort

Algoritma pengurutan yang diklasifikasikan kedalam kategori ini mencari tempat yang tepat untuk suatu elemen data yang telah diketahui kedalam subkumpulan data yang telah terurut, kemudian melakukan penyisipan (*insertion*) data di tempat yang tepat tersebut.

(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.

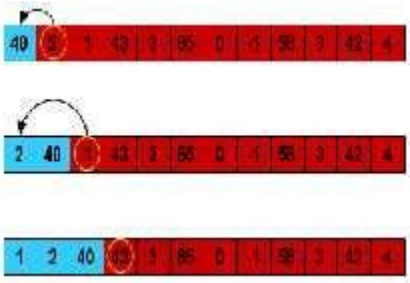
(2) *Implementasi Insertion sort*

a. Kondisi awal:
   - *Unsorted list* = data
   - *Sorted list* = kosong

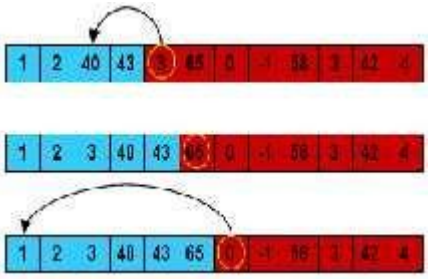
b. Ambil sembarang elemen dari *unsorted list*, sisipkan (*insert*) pada posisi yang benar dalam

sorted list. Lakukan terus sampai *unsorted list* habis.

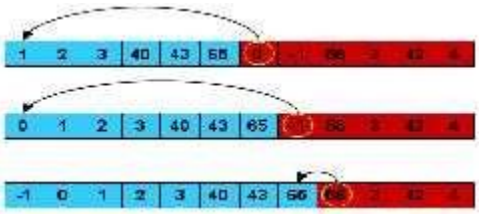
- **Langkah 1:**



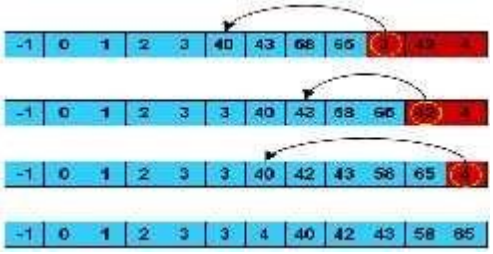
- **Langkah 2:**



- **Langkah 3:**



- **Langkah 4:**



2.2.5 Percobaan II-1: Bubble Sort

Ketik kode program di bawah ini:

```
#include <iostream>
using namespace std;

void tukar(int *a, int *b);

int main()
{
    int n,i,j;
```

```
    int arr[1005];

    cin >> n;
    for(i=0; i<n;i++)
        cin >> arr[i];

    for(i=0; i<n;i++)
        for(j=n-1; j>=i;j--)
            if(arr[i]>arr[j])
                tukar(&arr[i], &arr[j]);

    for(i=0; i<n;i++)
        cout << arr[i] << " ";
    return 0;
}

void tukar(int *a, int *b)
{
    int t=*a;
    *a=*b;
    *b=t;
}
```

Kode 2-1.bubbleSort.cpp

- Jelaskan kode program di atas!

2.2.6 Percobaan II-1: Bubble Sort

Ketik kode program di bawah ini:

```
#include <iostream>
using namespace std;

void tukar(int *a, int *b);

int main()
{
    int n,i,j;
    int arr[1005];

    cin >> n;
    for(i=0; i<n;i++)
        cin >> arr[i];

    for(i=0; i<n-1;i++)
        for(j=i+1; j<n;j++)
            if(arr[i]<arr[j])
                tukar(&arr[i], &arr[j]);
```

```
    for(i=0; i<n;i++)
        cout << arr[i] << " ";
    return 0;
}

void tukar(int *a, int *b)
{
    int t=*a;
    *a=*b;
    *b=t;
}
```

Kode 2-2.exchangeSort.cpp

- Jelaskan kode program di atas!

2.2.7 Percobaan II-2: Selection Sort

Ketik kode program di bawah ini:

```
#include <iostream>
using namespace std;

void tukar(int *a, int *b);

int main()
{
    int n,i,j, pos;
    int arr[1005];

    cin >> n;
    for(i=0; i<n;i++)
        cin >> arr[i];

    for(i=0; i<n-1;i++)
    {
        pos = i;
        for(j=i+1; j<n;j++)
            if(arr[j] > arr[pos])
                pos = j;
        if(pos != i)
            tukar(&arr[pos], &arr[i]);
    }
    for(i=0; i<n;i++)
        cout << arr[i] << " ";
    return 0;
}

void tukar(int *a, int *b)
{
```

```
    int t=*a;
    *a=*b;
    *b=t;
}
```

Kode 2-3. selectionSort.cpp

- Jelaskan kode program di atas

2.2.8 Percobaan II-3: Insertion Sort

Ketik kode program di bawah ini:

```
#include <iostream>
using namespace std;

void tukar(int *a, int *b);

int main()
{
    int n,i,j, temp;
    int arr[1005];

    cin >> n;
    for(i=0; i<n;i++)
        cin >> arr[i];

    for(i=1; i<n;i++)
    {
        temp = arr[i];
        j = i-1;
        while(arr[j]>temp && j>=0)
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = temp;
    }

    for(i=0; i<n;i++)
        cout << arr[i] << " ";
    return 0;
}

void tukar(int *a, int *b)
{
    int t=*a;
    *a=*b;
    *b=t;
}
```

Kode 2-4. insertionSort.cpp

- Jelaskan kode program di atas!

2.3 TUGAS AKHIR

- Buat flowchart dari Bubble Sort, Exchange Sort, Insertion Sort, dan Selection Sort!
- Buat program yang dapat memilih metode sorting yang mau diterapkan pada sejumlah data (jumlah data dinamis, input data dinamis)! Tampilkan proses yang terjadi di setiap tahapan dan berapa banyak iterasi yang harus dilakukan!

3 PERCOBAAN 3: SEARCHING

3.1 TUJUAN PERCOBAAN

- Memahami penggunaan searching
- Dapat memahami dan menggunakan searching

3.2 TINJAUAN PUSTAKA

Pada suatu data seringkali dibutuhkan pembacaan kembali informasi (*retrieval information*) dengan cara searching. Searching adalah pencarian data dengan cara menelusuri data-data tersebut. Tempat pencarian data dapat berupa array dalam memori, bisa juga pada *file* pada *external storage*.

Algoritma s*earching* yang umum terdiri dari 2 macam, yaitu:

- Sequential searching
- Binary searching

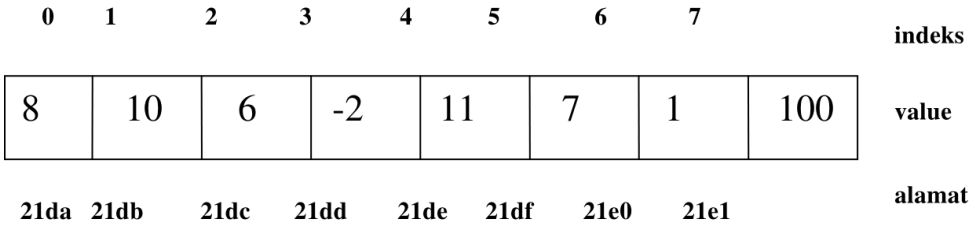
3.2.1 Sequential searching

Sequential searching adalah suatu teknik pencarian data dalam array (1 dimensi) yang akan menelusuri semua elemen-elemen array dari awal sampai akhir, dimana data-data tidak perlu diurutkan terlebih dahulu.

Kemungkinan terbaik (best case) adalah jika data yang dicari terletak di indeks array terdepan (elemen array pertama) sehingga waktu yang dibutuhkan untuk pencarian data sangat sebentar (minimal).

Kemungkinan terburuk (worst case) adalah jika data yang dicari terletak di indeks array terakhir (elemen array terakhir) sehingga waktu yang dibutuhkan untuk pencarian data sangat lama (maksimal).

Misalnya terdapat array satu dimensi sebagai berikut:



Gambar 3-1. Array 1 Dimensi dengan 8 data

Kemudian program akan meminta data yang akan dicari, misalnya 6. Jika ada maka akan ditampilkan tulisan "ADA", sedangkan jika tidak ada maka akan ditampilkan tulisan "TIDAK ADA".

Prinsip pencarian adalah:

1. Program menggunakan sebuah variabel flag yang berguna untuk menandai ada atau tidaknya data yang dicari dalam array data. Hanya bernilai 0 atau 1.
2. Flag pertama kali diinisialiasasi dengan nilai 0. • Jika ditemukan, maka flag akan diset menjadi 1, jika tidak ada maka flag akan tetap bernilai 0.
3. Semua elemen array data akan dibandingkan satu persatu dengan data yang dicari dan diinputkan oleh user.

3.2.2 Binary searching

Data yang ada harus diurutkan terlebih dahulu berdasarkan suatu urutan tertentu yang dijadikan kunci pencarian. Adalah teknik pencarian data dalam dengan cara membagi data menjadi dua bagian setiap kali terjadi proses pencarian.

Prinsip pencarian biner adalah:

1. Data diambil dari posisi 1 sampai posisi akhir N
2. Kemudian cari posisi data tengah dengan rumus: (posisi awal + posisi akhir) / 2
3. Kemudian data yang dicari dibandingkan dengan data yang di tengah, apakah sama atau lebih kecil, atau lebih besar?
4. Jika lebih besar, maka proses pencarian dicari dengan posisi awal adalah posisi tengah + 1
5. Jika lebih kecil, maka proses pencarian dicari dengan posisi akhir adalah posisi tengah – 1
6. Jika data sama, berarti ketemu.

3.3 PERCOBAAN

3.3.1 Percobaan 3-1: Sequential Searching

Ketik kode program di bawah ini:

```
#include <iostream>
using namespace std;

int main()
{
    int n = 8;
    int data[n] = {3, 6, 3, 8, 5, 9, 1, 4};
    int cari = 5;
    int i = 0;
    int flag = 0;
    while(i<n) {
        if(data[i] == cari) {
            flag = 1;
            break;
        }
        i++;
    }
```

```
    if(flag == 1)
        cout << "ketemu" << endl;
    else
        cout << "tidak ketemu" << endl;
    return 0;
}
```

Kode 3-1. sequentialSearching.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int n = 100;
    int data[n];
    int cari = 28;
    int counter = 0;
    int flag = 0;
    for(int i=0; i < n; i++) {
        data[i] = rand() % 100 + 1;
        cout << data[i] << " ";
    }
    cout << endl;

    for(int i=0; i < n; i++) {
        if(data[i] == cari) {
            counter++;
            flag = 1;
        }
    }

    if(flag == 1)
        cout << "Ketemu, sebanyak " << counter << endl;
    else
        cout << "tidak ketemu" << endl;
    return 0;
}
```

Kode 3-2. sequentialSearchingCounter.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int n = 10;
    int data[n] = { 7, 5, 3, 8, 5, 9, 7, 4};
    int cari = 9;
```

```
    int i = 0;
    data[5] = cari;
    while(data[i] != cari) {
        i++;
    }

    if(i < 5)
        cout << "Ketemu" << endl;
    else
        cout << "Tidak ketemu" << endl;
    return 0;
}
```

Kode 3-3. sequentialSearchingSentinel.cpp

- Jelaskan kode program di atas!

3.3.2 Percobaan 3-2: Binary Searching

Ketik kode program di bawah ini:

```
#include <iostream>
using namespace std;

int main()
{
    int n = 10;
    int data[n] = { 2, 5, 7, 8, 9, 12, 14, 24};
    int l, r, m;
    l = 0;
    r = n-1;
    int flag = 0;
    int cari = 12;

    while(l <= r && flag == 0) {
        m = (l+r)/2;
        cout << "data tengah: " << m << endl;
        if (data[m] == cari)
            flag = 1;
        else if (cari < data[m]) {
            cout << "cari di kiri" << endl;
            r = m-1;
        }
        else {
            cout << "cari di kanan" << endl;
            l = m+1;
        }
    }

    if(flag == 1)
```

```
        cout << "Ketemu" << endl;
    else
        cout << "Tidak ketemu" << endl;
    return 0;
}
```

Kode 3-4. binarySearching.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int n = 10;
    int data[n] = { 2, 15, 27, 38, 39, 42, 54, 74, 81, 105};
    int low, high;
    low = 0;
    high = n-1;
    int pos;
    int cari = 81;

    while(cari > data[low] && cari <= data[high]) {
        pos = (cari-data[low])/(data[high]-data[low]) * (high-low) + low;
        cout << pos << endl;
        if(cari > data[pos])
            low = pos+1;
        else if (cari < data[pos])
            high = pos-1;
        else
            low = pos;
    }

    if(cari == data[low])
        cout << "Ketemu" << endl;
    else
        cout << "Tidak ketemu" << endl;
    return 0;
}
```

Kode 3-5. binarySearchingInterpolation.cpp

- Jelaskan kode program di atas!

3.4 TUGAS AKHIR

- Buat flowchart dari sequential searching dan binary searching!
- Buat program yang menerima masukan data dinamis dan kemudian lakukan pencarian menggunakan sequential searching serta binary searching! Tampilkan jumlah iterasi dan tahapan di setiap iterasinya!

4 PERCOBAAN 4: LINKED LIST

4.1 TUJUAN PERCOBAAN

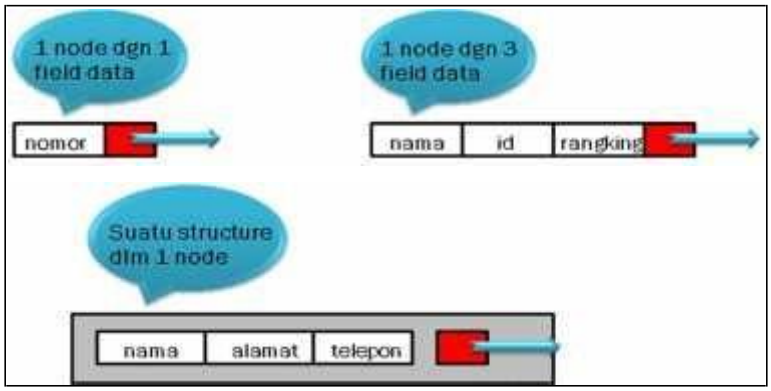
- Memahami Linked List
- Dapat membuat Linked List

4.2 TINJAUAN PUSTAKA

Linked list dikembangkan tahun 1955-1956 oleh Allen Newell, Cliff Shaw & Herbert Simon di RAND *Corporation* sebagai struktur data utama untuk bahasa *information processing language* (IPL). IPL adalah pengembangan dari program AI dengan bahasa pemrogaman COMMIT. *Linked list* merupakan struktur data dinamis yang paling sederhana yang berlawanan dengan *array*, yang merupakan struktur statis.

Linked list adalah koleksi dari obyek-obyek homogen dengan sifat setiap obyek (kecuali terakhir) punya penerus dan setiap obyek (kecuali yang pertama) punya pendahulu. Masing-masing data dalam *Linked list* disebut dengan *node* (simpul) yang menempati alokasi memori secara dinamis dan biasanya berupa *struct* yang terdiri dari beberapa *field*. Kumpulan komponen obyek- obyek ini disusun secara berurutan dengan bantuan pointer. Penggunaan pointer untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walau tidak bersebelahan secara fisik di memori. Masing-masing *linked list* terbagi menjadi 2 bagian :

- Medan informasi. Berisi informasi yang akan disimpan dan diolah.
- Medan penyambung. Berisi alamat simpul berikutnya



Gambar 4-1. Ilustrasi Linked List

Operasi pada *Linked list*

1. **Menambah simpul.** Bisa dipecahkan berdasarkan posisi simpul yaitu simpul baru:
   a. ditambahkan dibelakang simpul terakhir
   b. selalu diletakkan sebagai simpul pertama
2. menyisip diantara dua simpul yang sudah ada.**Menghapus simpul.** Dapat dilakukan dengan menghapus didepan, dibelakang atau ditengah dan harus berada sesudah simpul yang ditunjuk oleh suatu pointer.

3. **Mencari Informasi pada suatu *Linked list*.** Sama dengan membaca isi simpul hanya ditambah dengan test untuk menentukan ada tidaknya data yang dicari
4. **Mencetak Simpul.** Dengan cara membaca secara maju dan secara mundur.

4.2.1 Single Linked List

Linked list dapat diilustrasikan seperti satu kesatuan rangkaian kereta api. Kereta api terdiri dari beberapa gerbong, masing-masing dari gerbong itulah yang disebut node/tipe data bentukan. Agar gerbong-gerbong tersebut dapat saling bertaut dibutuhkan minimal sebuah kait yang dinamakan sebagai pointer. Setelah mendeklarasikan tipe data dan pointer pada list, selanjutnya kita akan mencoba membuat senarai / *Linked list* tunggal tidak berputar atau sebuah gerbong.

Single *Linked list* adalah senarai berkait yang masing-masing simpul pembentuknya mempunyai satu kait (link) ke simpul lainnya. Pembentukan *linked list* tunggal memerlukan:

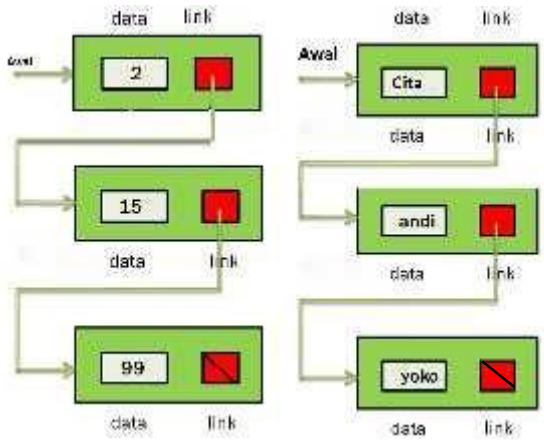
1. deklarasi tipe simpul
2. deklarasi variabel pointer penunjuk awal *Linked list*
3. pembentukan simpul baru
4. pengaitan simpul baru ke *Linked list* yang telah terbentuk

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

Gambar 4-2. Ilustrasi Single Linked List

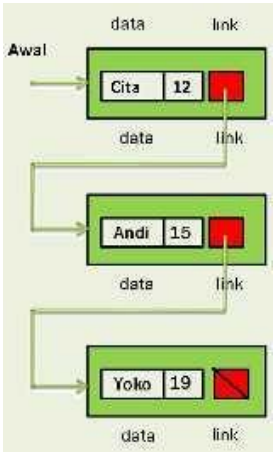
Gambar 4-2 mengilustrasikan sebuah rangkaian kereta api dengan 4 buah gerbong. Gerbong A akan disebut sebagai kepala / head (walaupun penamaan ini bebas) dan gerbong D adalah ekor / tail. Tanda panah merupakan kait atau pointer yang menghubungkan satu gerbong dengan yang lainnya.

Pointer yang dimiliki D menuju ke NULL, inilah yang membedakan antara senarai berputar dengan yang tidak berputar. Kalau senarai berputar maka pointer dari D akan menuju ke A lagi.



Gambar 4-3. Linked list dengan medan informasi berisi 1 field data

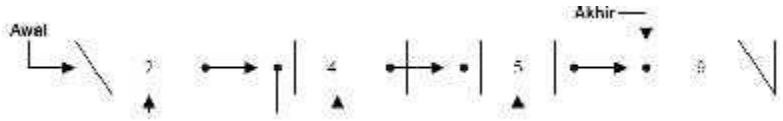
Contoh deklarasi single linked list dengan 1 node berisi 2 field data:



Gambar 4-4. Linked list dengan medan informasi berisi 2 field data

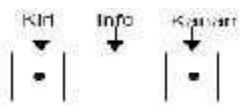
4.2.2 Double Linked List

Double linked list adalah suatu *Linked list* yang mempunyai 2 penunjuk yaitu penunjuk ke data sebelumnya dan berikutnya, seperti terlihat pada Gambar 20.



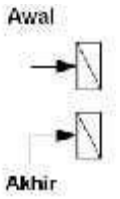
Gambar 4-5. Ilustrasi double linked list

Jika dilihat 1 elemen listnya, maka secara umum struktur dari elemen listnya adalah sebagai berikut :



Gambar 4-6. Dekripsi simpul double linked list

Dari Gambar 4-6 untuk setiap elemen terdiri dari 3 buah field yaitu kiri (prev), info (data), dan kanan (next). Field kiri dan kanan merupakan sebuah pointer ke data struktur elemen (tdata).



Gambar 4-7. Dekripsi simpul double linked list setelah dideklarasikan

4.3 PERCOBAAN

4.3.1 Percobaan 4-1: Linked List Beginning Insertion

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;

struct node
{
    int info;
    node *next;
} *start, *newptr, *save, *ptr;

node *create_new_node(int);
void insert_at_beg(node *);
void display(node *);

int main()
{
    start = NULL;
    int inf;
    char ch='y';

    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan nilai untuk node baru: ";
        cin>>inf;
        cout<<"Membuat node baru..." << endl;

        newptr = create_new_node(inf);
        if(newptr != NULL)
        {
```

```
            cout<<"Berhasil membuat node baru..." << endl;
                    }
        else
        {
            cout<<"Maaf, tidak dapat membuat node baru";
            return 0;
        }
        cout<<"Memasukkan node pada bagian awal list..." << endl;
        insert_at_beg(newptr);
        cout<<"Node berhasil dimasukkan di bagian awal list..." << endl;
        cout<<"List: ";
        display(start);
        cout<<"Mau membuat node baru? (y/n) ";
        cin>>ch;
    }

    return 0;
}

node *create_new_node(int n)
{
    ptr = new node;
    ptr->info = n;
    ptr->next = NULL;
    return ptr;
}

void insert_at_beg(node *np)
{
    if(start==NULL)
    {
        start = np;
    }
    else
    {
        save = start;
        start = np;
        np->next = save;
    }
}

void display(node *np)
{
    while(np != NULL)
    {
        cout<<np->info<<" -> ";
```

```
        np = np->next;
    }
    cout<<"!!\n";
}
```

Kode 4-1. linkedListBeginningInsertion.cpp

- Jelaskan kode program di atas!

4.3.2 Percobaan 4-2: Linked List End Insertion

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;

struct node
{
    int info;
    node *next;
} *start, *newptr, *save, *ptr, *rear;

node *create_new_node(int);
void insert_in_end(node *);
void display(node *);

int main()
{
    start = rear = NULL;
    int inf;
    char ch='y';

    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan nilai untuk node baru: ";
        cin>>inf;
        cout<<"Membuat node baru..." << endl;

        newptr = create_new_node(inf);
        if(newptr != NULL)
        {
            cout<<"Berhasil membuat node baru..." << endl;
                    }
        else
        {
            cout<<"Maaf, tidak dapat membuat node baru";
            return 0;
        }
        cout<<"Memasukkan node pada bagian akhir list..." << endl;
        insert_in_end(newptr);
```

```
        cout<<"Node berhasil dimasukkan di bagian akhir list..." << endl;
        cout<<"List: ";
        display(start);
        cout<<"Mau membuat node baru? (y/n) ";
        cin>>ch;
    }
    return 0;
}

node *create_new_node(int n)
{
    ptr = new node;
    ptr->info = n;
    ptr->next = NULL;
    return ptr;
}

void insert_in_end(node *np)
{
    if(start==NULL)
    {
        start = rear = np;
    }
    else
    {
        rear -> next = np;
        rear = np;
    }
}

void display(node *np)
{
    while(np != NULL)
    {
        cout<<np->info<<" -> ";
        np = np->next;
    }
    cout<<"!!\n";
}
```

Kode 4-2. linkedListEndInsertion.cpp

- Jelaskan kode program di atas!

4.3.3 Percobaan 4-3: Linked List Delete Node

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;
```

```
struct node
{
    int info;
    node *next;
} *start, *newptr, *save, *ptr, *rear;

node *create_new_node(int);
void insert_node(node *);
void display(node *);
void delete_node();

int main()
{
    start = rear = NULL;
    int inf;
    char ch='y';

    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan nilai untuk node baru: ";
        cin>>inf;
        cout<<"Membuat node baru..." << endl;

        newptr = create_new_node(inf);
        if(newptr != NULL)
        {
            cout<<"Berhasil membuat node baru..." << endl;
                    }
        else
        {
            cout<<"Maaf, tidak dapat membuat node baru";
            return 0;
        }
        cout<<"Memasukkan node pada bagian akhir list..." << endl;
        insert_node(newptr);
        cout<<"Node berhasil dimasukkan di bagian akhir list..." << endl;
        cout<<"List: ";
        display(start);
        cout<<"Mau membuat node baru? (y/n) ";
        cin>>ch;
    }

    do
    {
        cout<<"List:";
```

```
        display(start);
        cout<<"Mau menghapus node pertama? (y/n) ";
        cin>>ch;
        if(ch=='y' || ch=='Y');
        {
            delete_node();
        }
    }while(ch=='y' || ch=='Y');

    return 0;
}

node *create_new_node(int n)
{
    ptr = new node;
    ptr->info = n;
    ptr->next = NULL;
    return ptr;
}

void insert_node(node *np)
{
    if(start==NULL)
    {
        start = rear = np;
    }
    else
    {
        rear -> next = np;
        rear = np;
    }
}

void delete_node()
{
    if(start == NULL)
    {
        cout<<"Underflow...!!" << endl;
    }
    else
    {
        ptr = start;
        start = start->next;
    }
}
```

```
void display(node *np)
{
    while(np != NULL)
    {
        cout<<np->info<<" -> ";
        np = np->next;
    }
    cout<<"!!\n";
}
```

Kode 4-3. linkedListDelete.cpp

- Jelaskan kode program di atas!

4.3.4 Percobaan 4-4: Linked List Traversal

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;

struct node
{
    int info;
    node *next;
} *start, *newptr, *save, *ptr, *rear;

node *create_new_node(int);
void insert_node(node *);
void travers(node *);

int main()
{
    start = rear = NULL;
    int inf;
    char ch='y';

    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan nilai untuk node baru: ";
        cin>>inf;
        cout<<"Membuat node baru..." << endl;

        newptr = create_new_node(inf);
        if(newptr != NULL)
        {
            cout<<"Berhasil membuat node baru..." << endl;
                    }
        else
        {
```

```
            cout<<"Maaf, tidak dapat membuat node baru";
            return 0;
        }
        cout<<"Memasukkan node pada bagian akhir list..." << endl;
        insert_node(newptr);
        cout<<"Node berhasil dimasukkan di bagian akhir list..." << endl;

        cout<<"Mau membuat node baru? (y/n) ";
        cin>>ch;
    }

    cout<<"List: ";
    travers(start);

    if(start != NULL)
    {
        cout<<"Mengakses nilai pada satu node: ";
        cout<< start->info << endl;
        cout<<"Alamat dari node ini: ";
        cout<< &start << endl;
        cout<<"Alamat dari node selanjutnya: ";
        cout<< start->next << endl;
    }
    if(start->next != NULL)
    {
        cout<<"Mengakses nilai pada node selanjutnya: ";
        cout<< start->next->info << endl;
        cout<<"Alamat dari node ini: ";
        cout<< start->next << endl;
        cout<<"Alamat dari node selanjutnya: ";
        cout<< start->next->next << endl;
    }

    return 0;
}

node *create_new_node(int n)
{
    ptr = new node;
    ptr->info = n;
    ptr->next = NULL;
    return ptr;
}

void insert_node(node *np)
{
```

```
    if(start==NULL)
    {
        start = rear = np;
    }
    else
    {
        rear -> next = np;
        rear = np;
    }
}

void travers(node *np)
{
    while(np != NULL)
    {
        cout<<np->info<<" -> ";
        np = np->next;
    }
    cout<<" !!\n";
}
```

Kode 4-4. linkedListEndInsertion.cpp

- Jelaskan kode program di atas!

4.4 TUGAS AKHIR

- Buat flowchart untuk pembuatan Linked List!
- Buat program untuk Double Linked List!
- Buat program yang menampung 5 field data menggunakan Linked List!

5 PERCOBAAN 5: STACK

5.1 TUJUAN PERCOBAAN

- Memahami bagaimana Stack Bekerja
- Dapat membuat Stack

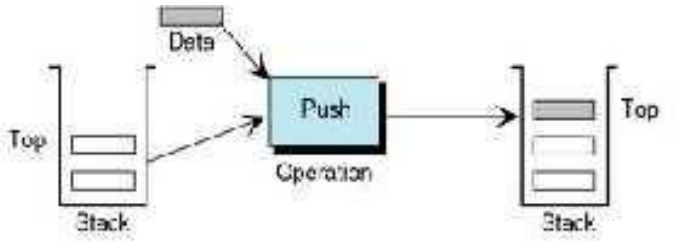
5.2 TINJAUAN PUSTAKA

Stack bisa diartikan sebagai suatu kumpulan data yang seolah-olah ada data yang diletakkan di atas data yang lain. Di dalam *stack*, dapat dilakukan penambahan/ penyisipan dan pengambilan/penghapusan data melalui ujung yang sama yang disebut sebagai puncak *stack* (*top of stack*). *Stack* mempunyai sifat LIFO (*Last In First Out*), artinya data yang terakhir masuk adalah data yang pertama keluar.

Contoh dalam kehidupan sehari-hari adalah tumpukan piring di sebuah restoran yang tumpukannya dapat ditambah pada bagian paling atas dan jika mengambilnya pun dari bagian paling atas pula.

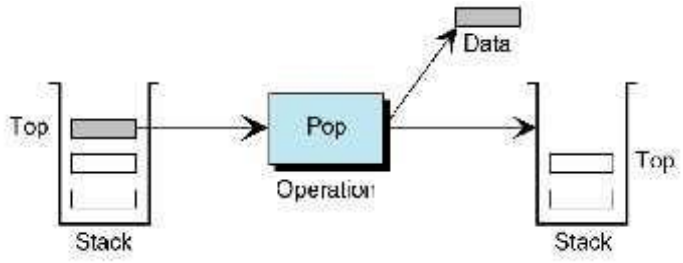
Ada 2 operasi paling dasar dari *stack* yang dapat dilakukan, yaitu :

1. Operasi ***push.*** yaitu operasi menambahkan elemen pada urutan terakhir (paling atas). Dengan syarat tumpukan tidak dalam kondisi penuh, jika penuh penuh maka terjadi *overflow*.



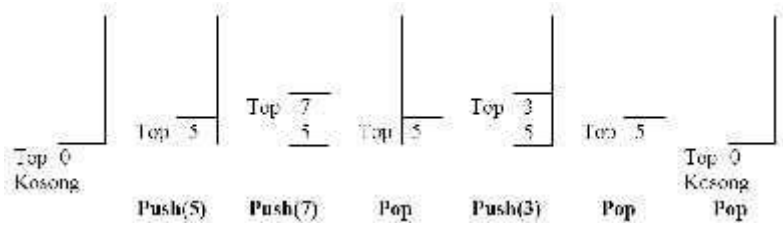
Gambar 5-1. Operasi Push pada Stack

2. Operasi **pop.** yaitu operasi mengambil sebuah elemen data pada urutan terakhir dan menghapus elemen tersebut dari *stack*. Dengan syarat tumpukan tidak dalam kondisi kosong, jika kosong akan terjadi *underflow*.



Gambar 5-2. Operasi Pop pada Stack

Contoh: ada sekumpulan perintah *stack* yaitu *push*(5), *push*(7), pop, *push*(3), pop. Jika dijalankan, maka yang akan terjadi adalah:



Gambar 5-3. Contoh operasi dasar pada stack

Selain operasi dasar *stack* (*push* dan *stack*), ada lagi operasi lain yang dapat terjadi dalam *stack* yaitu :

- Proses **deklarasi** yaitu proses pendeklarasian *stack*.
- Proses ***IsEmpty*** yaitu proses pemeriksaan apakah *stack* dalam keadaan kosong.
- Proses **IsFull** yaitu proses pemeriksaan apakah *stack* telah penuh.
- Proses **inisialisasi** yaitu proses pembuatan *stack* kosong, biasanya dengan pemberian nilai untuk top.

Representasi *stack* dalam pemrograman, dapat dilakukan dengan 2 cara yaitu :

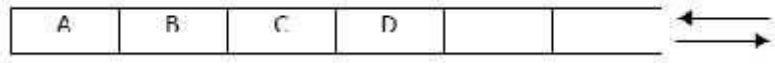
- Representasi *stack* dengan array
- Representasi *stack* dengan single linked list

5.2.1 Representasi Stack Dengan Array

Bentuk penyajian *stack* menggunakan tipe data array sebenarnya kurang tepat karena banyaknya elemen dalam array bersifat statis, sedangkan jumlah elemen *stack* sangat bervariasi atau dinamis. Meskipun demikian, array dapat digunakan untuk penyajian *stack*, tetapi dengan anggapan bahwa banyaknya elemen maksimal suatu *stack* tidak melebihi batas maksimum banyaknya array. Pada suatu saat ukuran *stack* akan sama dengan ukuran array. Bila diteruskan menambah data, maka akan terjadi *overflow*. Oleh karena itu, perlu ditambahkan data untuk mencatat posisi ujung *stack*. Ada dua jenis bentuk *stack* menggunakan array, yaitu single *stack* dan double *stack*.

5.2.1.1 Single Stack

Single *stack* dapat dianalogikan dengan sebuah wadah yang diisi benda melalui satu jalan keluar dan masuk.



Gambar 5-4. Ilustrasi Single Stack

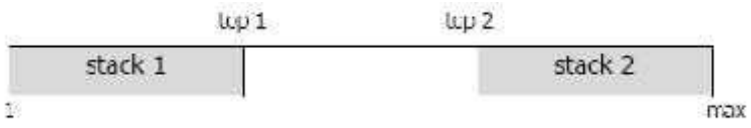
Representasi *stack* dengan menggunakan array dengan maksimal data 9 dapat dilihat pada Gambar 5-5.

28
45
27
70
60
50
15
25
30
5

Gambar 5-5. Representasi Stack menggunakan array dengan 9 data

5.2.1.2 Double Stack

Double stack merupakan bentuk pengembangan single *stack* dengan maksud untuk menghemat memori. Dalam *double stack* terdapat dua *stack* dalam satu array. *Stack* 1 bergerak ke kanan dan *stack* 2 bergerak ke kiri. *Double stack* dikatakan penuh apabila puncak *stack* 1 bertemu dengan puncak *stack* 2.

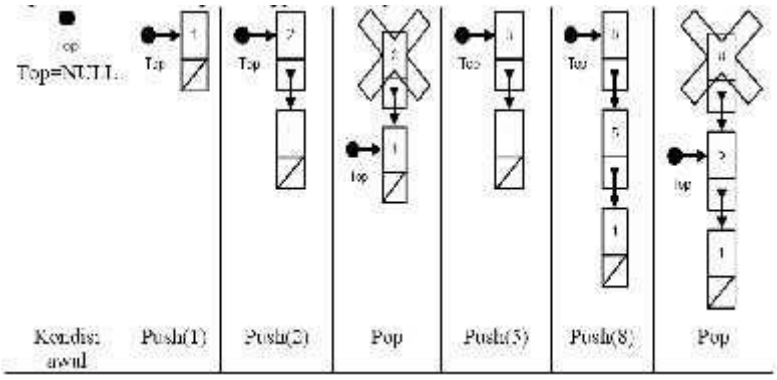


Gambar 5-6. Ilustrasi double stack

5.2.2 Representasi Stack Dengan Single Linked List

Bentuk penyajian *stack* menggunakan single linked list sangat tepat karena banyaknya elemen dalam list bersifat dinamis, sedangkan jumlah elemen *stack* juga sangat bervariasi atau dinamis.

Representasi *stack* mengunakan single linkd list dapat dilihat pada Gambar 5-7.



Gambar 5-7. Representasi stack menggunakan single linked list

5.3 PERCOBAAN

5.3.1 Percobaan 5-1: Single Stack dengan Struct

Ketik kode program di bawah ini:

```
#include<iostream>
#include <cstring>
using namespace std;
#define MAX_STACK 10

struct STACK
{
    int top;
    char data[10][10];
};
STACK tumpuk;

void inisialisasi();
int isFull();
int isEmpty();
void push(char d[10]);
void pop();
void clear();
void tampilStack();

int main()
{
    int pil;
    inisialisasi();
    char dt[10];
    do{
        cout << "1. push" << endl;
        cout << "2. pop" << endl;
        cout << "3. print" << endl;
        cout << "4. clear" << endl;
        cout << "5. exit" << endl;
        cout << "Pilihan : ";
        cin >> pil;
        switch(pil){
            case 1:
                if(isFull() != 1){
                    cout << "Data = ";
                    cin >> dt;
                    push(dt);
                }
                else
                    cout << "Sudah penuh!" << endl;
```

```
            break;
            case 2:
                if(isEmpty() != 1)
                    pop();
                else
                    cout << "Masih kosong!" << endl;
            break;
            case 3:
                if(isEmpty() != 1)
                    tampilStack();
                else
                    cout << "Masih kosong!" << endl;
            break;
            case 4:
                clear();
                cout << "Sudah kosong!" << endl;
            break;
        }

    }while(pil != 5);

    return 0;
}

void inisialisasi()
{
    tumpuk.top = -1;
}
int isFull()
{
    if(tumpuk.top == MAX_STACK-1)
        return 1;
    else
        return 0;
}
int isEmpty()
{
    if(tumpuk.top == -1)
        return 1;
    else
        return 0;
}
void push(char d[10])
{
    tumpuk.top++;
    strcpy(tumpuk.data[tumpuk.top],d);
```

```
}
void pop()
{
    cout << "Data yang terambil = " << tumpuk.data[tumpuk.top] << endl;
    tumpuk.top--;
}
void clear()
{
    tumpuk.top=-1;
}

void tampilStack()
{
    for(int i=tumpuk.top;i>=0;i--)
    {
        cout << "Data : " << tumpuk.data[i] << endl;
    }
}
```

Kode 5-1. stackStruct.cpp

- Jelaskan kode program di atas!

5.3.2 Percobaan 5-2: Stack dengan Array Push

Ketik kode program di bawah ini:

```
#include<iostream>
#include <cstring>
using namespace std;
#define MAX_STACK 10

int push(int [], int &, int);
void display(int [], int);
const int SIZE = 50;

int main()
{
    int stack[SIZE], item, top=-1, res;
    char ch='y';
    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan elemen: ";
        cin>>item;
        res = push(stack, top, item);
        if(res == -1)
        {
            cout<<"Overflow..!!..Keluar program..!!";
            return 0;
        }
```

```
        cout<<"Elemen berhasil dimasukkan" << endl;
        cout<<"Stack: ";
        display(stack, top);
        cout<<"Mau menambahkan elemen ? (y/n) ";
        cin>>ch;
    }
    return 0;
}

int push(int stack[], int &top, int elem)
{
    if(top == SIZE-1)
    {
        return -1;
    }
    else
    {
        top++;
        stack[top] = elem;
    }
    return 0;
}
void display(int stack[], int top)
{
    cout<<stack[top]<<" <-- "<<"\n";
    for(int i=top-1; i>=0; i--)
    {
        cout<<stack[i]<<"\n";
    }
}
```

Kode 5-2. stackArrayPush.cpp

- Jelaskan kode program di atas!

5.3.3 Percobaan 5-3: Stack dengan Array Pop

Ketik kode program di bawah ini:

```
#include<iostream>
#include <cstring>
using namespace std;
#define MAX_STACK 10

int pop(int [], int &);
int push(int [], int &, int);
void display(int [], int);
const int SIZE = 50;

int main()
```

```
{
    int stack[SIZE], item, top=-1, res;
    char ch='y';
    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan elemen: ";
        cin>>item;
        res = push(stack, top, item);
        if(res == -1)
        {
            cout<<"Overflow..!!..Keluar program..!!";
            return 0;
        }
        cout<<"Elemen berhasil dimasukkan" << endl;
        cout<<"Stack: ";
        display(stack, top);
        cout<<"Mau menambahkan elemen? (y/n) ";
        cin>>ch;
    }

    cout << "Penghapusan elemen dimulai..." << endl;
    ch='y';
    while(ch=='y' || ch=='Y')
    {
        res = pop(stack, top);
        if(res==-1)
        {
            cout<<"Underflow..!!..Aborting..!!..Keluar program...";
            return 0;
        }
        else
        {
            cout<<"Elemen yang dihapus adalah: "<<res<<endl;
            cout<<"Stack: ";
            display(stack, top);
        }
        cout<<"Mau menghapus lagi? (y/n).. ";
        cin>>ch;
    }
    return 0;
}

int push(int stack[], int &top, int elem)
{
    if(top == SIZE-1)
    {
```

```
        return -1;
    }
    else
    {
        top++;
        stack[top] = elem;
    }
    return 0;
}

int pop(int stack[], int &top)
{
    int ret;
    if(top==-1)
    {
        return -1;
    }
    else
    {
        ret=stack[top];
        top--;
    }
    return ret;
}

void display(int stack[], int top)
{
    cout<<stack[top]<<" <-- "<<"\n";
    for(int i=top-1; i>=0; i--)
    {
        cout<<stack[i]<<"\n";
    }
}
```

Kode 5-3. stackArrayPop.cpp

- Jelaskan kode program di atas!

5.3.4 Percobaan 5-4: Stack dengan Single Linked List

Ketik kode program di bawah ini:

```
#include<iostream>
#include <cstring>
using namespace std;

struct node
{
    int info;
    node *next;
```

```
} *top, *newptr, *save, *ptr;

node *create_new_node(int);
void push(node *);
void display(node *);

int main()
{
    int inf;
    char ch='y';
    top=NULL;
    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan elemen: ";
        cin>>inf;
        newptr = create_new_node(inf);
        if(newptr == NULL)
        {
            cout<<"Maaf, tidak bisa membuat node..Keluar program..!!";
            return 0;
        }
        cout<<"Elemen dimasukkan..." << endl;
        push(newptr);
        cout<<"Elemen berhasil dimasukkan..." << endl;
        cout<<"Stack: ";
        display(top);
        cout<<"Mau menambahkan elemen ? (y/n) ";
        cin>>ch;
    }
    return 0;
}

node *create_new_node(int x)
{
    ptr = new node;
    ptr->info = x;
    ptr->next = NULL;
    return ptr;
}

void push(node *n)
{
    if(top==NULL)
    {
        top=n;
    }
```

```
    else
    {
        save = top;
        top = n;
        n->next = save;
    }
}

void display(node *n)
{
    while(n != NULL)
    {
        cout<<n->info<<" -> ";
        n = n->next;
    }
    cout<<"!!\n";
}
```

Kode 5-4. StackLinkedListPush.cpp

- Jelaskan kode program di atas!

5.3.5 Percobaan 5-5: Stack dengan Single Linked List Pop

Ketik kode program di bawah ini:

```
#include<iostream>
#include <cstring>
using namespace std;

struct node
{
    int info;
    node *next;
} *top, *newptr, *save, *ptr;

node *create_new_node(int);
void push(node *);
void pop();
void display(node *);

int main()
{
    int inf;
    char ch='y';
    top=NULL;
    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan elemen: ";
        cin>>inf;
```

```
        newptr = create_new_node(inf);
        if(newptr == NULL)
        {
            cout<<"Maaf, tidak bisa membuat node..Keluar program..!!";
            return 0;
        }
        cout<<"Elemen dimasukkan..." << endl;
        push(newptr);
        cout<<"Elemen berhasil dimasukkan..." << endl;
        cout<<"Stack: ";
        display(top);
        cout<<"Mau menambahkan elemen ? (y/n) ";
        cin>>ch;
    }

    do
    {
        cout<<"Stack: \n";
        display(top);
        cout<<"Mau mengeluarkan elemen? (y/n) ";
        cin>>ch;
        if(ch=='y' || ch=='Y')
        {
            cout << "Elemen yang dikeluarkan: " << top->info << endl;
            pop();
        }
        cout<< endl;
    }while(ch=='y' || ch=='Y');

    return 0;
}

node *create_new_node(int x)
{
    ptr = new node;
    ptr->info = x;
    ptr->next = NULL;
    return ptr;
}

void push(node *n)
{
    if(top==NULL)
    {
        top=n;
    }
```

```
    else
    {
        save = top;
        top = n;
        n->next = save;
    }
}

void pop()
{
    if(top==NULL)
    {
        cout<<"Underflow..!!..Keluar dari program...";
        exit(1);
    }
    else
    {
        ptr = top;
        top = top->next;
    }
}

void display(node *n)
{
    while(n != NULL)
    {
        cout<<n->info<<" -> ";
        n = n->next;
    }
    cout<<"!!\n";
}
```

Kode 5-5. StackLinkedListPop.cpp

- Jelaskan kode program di atas!

5.4 TUGAS AKHIR

- Buat flowchart untuk single stack menggunakan array, double stack menggunakan array, dan single stack menggunakan single linked list!
- Buat program catatan harian menggunakan stack!

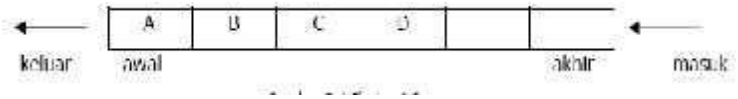
6 PERCOBAAN 6: QUEUE

6.1 TUJUAN PERCOBAAN

- Memahami Queue pada Struktur Data
- Dapat memahami dan menggunakan Queue

6.2 TINJAUAN PUSTAKA

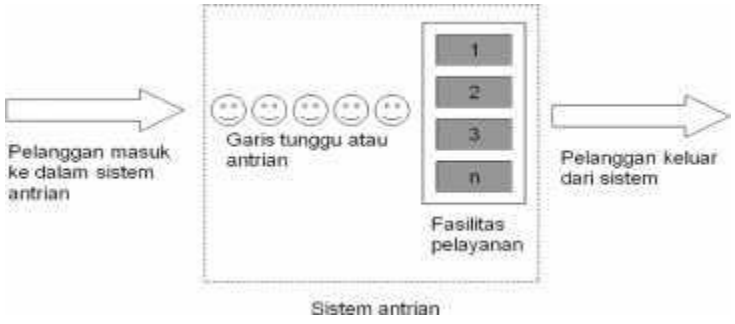
Secara harfiah *queue* dapat diartikan sebagai antrian. *Queue* merupakan kumpulan data dengan penambahan data hanya melalui satu sisi, yaitu belakang (*tail*) dan penghapusan data hanya melalui sisi depan (*head*). Berbeda dengan *stack* yang bersifat LIFO maka *queue* bersifat FIFO*(First In First Out)*, yaitu data yang pertama masuk akan keluar terlebih dahulu dan data yang terakhir masuk akan keluar terakhir. Berikut ini adalah gambaran struktur data *queue*.



Gambar 6-1 Ilustrasi queue

Elemen yang pertama kali masuk ke dalam *queue* disebut elemen depan (*front/head of queue*), sedangkan elemen yang terakhir kali masuk ke *queue* disebut elemen belakang (rear/ tail of *queue*).

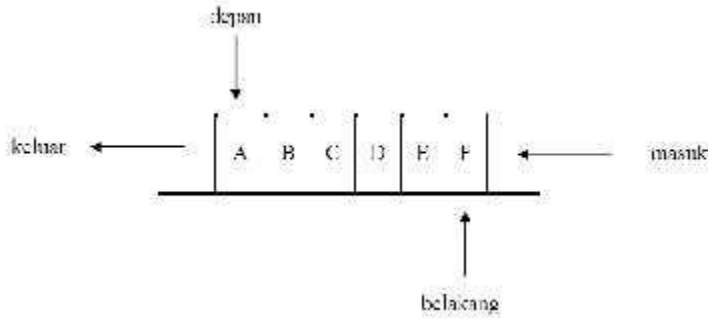
Perbedaan antara *stack* dan *queue* terdapat pada aturan penambahan dan penghapusan elemen. Pada *stack*, operasi penambahan dan penghapusan elemen dilakukan di satu ujung. Elemen yang terakhir kali dimasukkan akan berada paling dekat dengan ujung atau dianggap paling atas sehingga pada operasi penghapusan, elemen teratas tersebut akan dihapus paling awal, sifat demikian dikenal dengan LIFO. Pada *queue*, operasi tersebut dilakukan di tempat yang berbeda. Penambahan elemen selalu dilakukan melalui salah satu ujung, menempati posisi di belakang elemen-elemen yang sudah masuk sebelumnya atau menjadi elemen paling belakang. Sedangkan penghapusan elemen dilakukan di ujung yang berbeda, yaitu pada posisi elemen yang masuk paling awal atau elemen terdepan. Sifat yang demikian dikenal dengan FIFO.



Gambar 6-2. Contoh model antrian

6.2.1 Representasi Queue Dengan Array

Disebut juga *queue* dengan model fisik, yaitu bagian depan *queue* selalu menempati posisi pertama array.



Gambar 6-3. Contoh antrian dengan 6 elemen

6.2.2 Representasi Queue Dengan Linked List

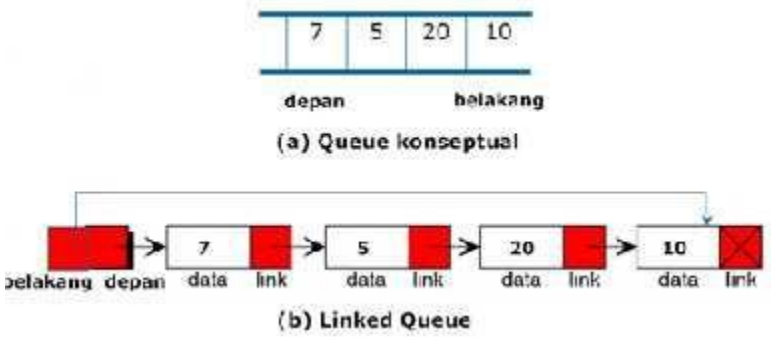
Implementasi antrian secara dinamis dilakukan dengan menggunakan tipe data *pointer*. Pada dasarnya sama dengan implementasi list pada bab III dimana pada *queue* dikenal sebagai sistem FIFO (first in first out).

Operasi yang umum pada *Queue*:

(1) ***Buat_queue.*** Mendeklarasikan *queue* yg kosong/menginisialisasi *queue* yg kosong.
(2) ***Enqueue.*** Menambah elemen pada posisi paling belakang.
(3) ***Dequeue.*** Mengeluarkan elemen pada posisi paling depan.
(4) ***Rear***. melihat elemen paling belakang dari antrian.
(5) ***Front***. Melihat elemen terdepan dari antrian.

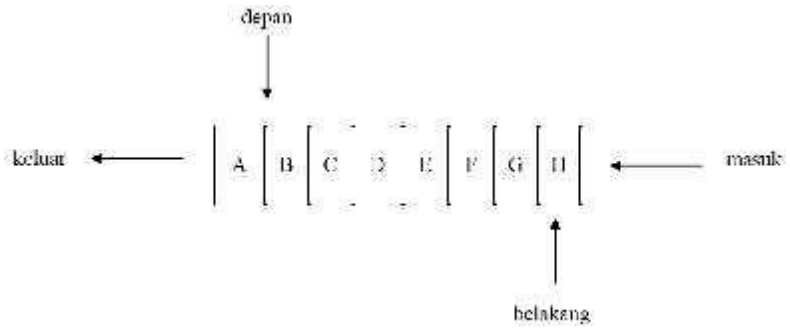
Elemen antrian disimpan sebagai node yang dibuat dengan memori dinamis.

- Tiap node memiliki field:
  - Data : berisi elemen data antrian
  - Link : pointer ke node berikut dalam antrian
- Structure suatu antrian memuat field-field berikut:
  - Depan : pointer ke node pertama dalam antrian
  - Belakang : pointer ke node terakhir dalam antrian



Gambar 6-4. Ilustrasi queue menggunakan linked list

Gambar 6-4 di atas menunjukkan contoh penyajian antrian menggunakan larik. Antrian di atas berisi 6 elemen, yaitu A,B,C,D,E dan F. Elemen A terletak di bagian depan antrian dan elemen F terletak dibagian belakang antrian. Dengan demikian, jika ada elemen yang baru masuk, maka ia akan diletakkan disebelah kanan F (pada gambar diatas). Jika ada elemen yang akan dihapus, maka A akan dihapus lebih dahulu. Gambar 6-5. menunjukkan antrian di atas setelah berturut-turut dimasukkan G dan H.



Gambar 6-5. Ilustrasi penambahan elemen pada antrian

6.3 PERCOBAAN

6.3.1 Percobaan 6-1: Queue dengan Struct

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;
#define MAX 8

typedef struct{
    int data[MAX];
    int head;
    int tail;
} Queue;
Queue antrian;

void create();
int isEmpty();
int isFull();
void enqueue(int data);
int dequeue();
void clear();
void tampil();

int main()
{
    int pil;
    int data;
    create();
```

```
    do{
        cout << "1. Enqueue" << endl;
        cout << "2. Dequeue" << endl;
        cout << "3. Tampil" << endl;
        cout << "4. Clear" << endl;
        cout << "5. Exit" << endl;
        cout << "Pilihan = ";
        cin >> pil;
        switch(pil){
            case 1:
            cout << "Data = ";
            cin >> data;
            enqueue(data);
            break;
            case 2:
            cout << "Elemen yang keluar : " << dequeue() << endl;
            break;
            case 3:
            tampil();
            break;
            case 4:
            clear();
            break;
        }
    } while(pil!=5);

    return 0;
}

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

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

int isFull()
{
    if(antrian.tail==MAX-1)
        return 1;
```

```
    else
        return 0;
}

void enqueue(int data)
{
    if(isEmpty()==1){
        antrian.head=antrian.tail=0;
        antrian.data[antrian.tail]=data;
        cout << antrian.data[antrian.tail] << " masuk!" << endl;
    } else
    if(isFull()==0){
        antrian.tail++;
        antrian.data[antrian.tail]=data;
        cout << antrian.data[antrian.tail] << " masuk!" << endl;
    }
}

int dequeue()
{
    int i;
    int e = antrian.data[antrian.head];
    for(i=antrian.head;i<=antrian.tail-1;i++){
        antrian.data[i] = antrian.data[i+1];
    }
    antrian.tail--;
    return e;
}

void clear()
{
    antrian.head=antrian.tail=-1;
    cout << "data clear" << endl;
}

void tampil()
{
    if(isEmpty()==0){
        for(int i=antrian.head;i<=antrian.tail;i++)
        {
            cout << antrian.data[i] << " ";
        }
        cout << endl;
    }else cout << "data kosong!" << endl;
}
```

Kode 6-1. QueueStruct.cpp

- Jelaskan kode program di atas!

6.3.2 Percobaan 6-2: Queue dengan Array Enqueue

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;

int enqueue(int [], int);
void display(int [], int, int);

const int SIZE = 50;

int queue[SIZE];
int front=-1;
int rear=-1;

int main()
{
    int item, check;
    char ch='y';

    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan elemen pada antrian= ";
        cin>>item;
        check = enqueue(queue, item);
        if(check == -1)
        {
            cout<<"Overflow..!!..Keluar program.." << endl;
            exit(1);
        }
        cout<<"Antrian berhasil ditambahkan..." << endl;
        cout<<"Queue (Depan...ke...Belakang):" << endl;
        display(queue, front, rear);
        cout<<"Mau menambahkan antrian? (y/n) ";
        cin>>ch;
    }

    return 0;
}

int enqueue(int queue[], int elem)
{
    if(rear == SIZE-1)
    {
        return -1;
```

```
    }
    else if(rear == -1)
    {
        front = rear = 0;
        queue[rear] = elem;
    }
    else
    {
        rear++;
        queue[rear] = elem;
    }
    return 0;
}

void display(int queue[], int front, int rear)
{
    if(front == -1)
    {
        return;
    }
    for(int i=front; i<rear; i++)
    {
        cout<<queue[i]<<" <- ";
    }
    cout<<queue[rear]<< endl;
}
```

Kode 6-2. QueueArrayEnqueue.cpp

- Jelaskan kode program di atas!

6.3.3 Percobaan 6-3: Queue dengan Array Dequeue

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;

int dequeue(int []);
int enqueue(int [], int);
void display(int [], int, int);

const int SIZE = 50;

int queue[SIZE];
int front=-1;
int rear=-1;

int main()
{
```

```
    int item, check;
    char ch='y';

    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan data pada antrian= ";
        cin>>item;
        check = enqueue(queue, item);
        if(check == -1)
        {
            cout<<"Overflow..!!..Keluar program.." << endl;
            exit(1);
        }
        cout<<"Antrian berhasil ditambahkan..." << endl;
        cout<<"Queue (Depan...ke...Belakang):" << endl;
        display(queue, front, rear);
        cout<<"Mau menambahkan antrian? (y/n) ";
        cin>>ch;
    }

    cout<<"Memulai pengeluaran elemen...\n";
    ch='y';
    while(ch=='y' || ch=='Y')
    {
        check = dequeue(queue);
        if(check == -1)
        {
            cout<<"Underflow..!!..Keluar program.." << endl;
            exit(2);
        }
        else
        {
            cout<<"Elemen yang dikeluarkan dari antrian: "<<check<< endl;
            cout<<"Queue (Depan...ke...Belakang):" << endl;
            display(queue, front, rear);
        }
        cout<<"Mau mengeluarkan elemen dari antrian lagi? (y/n) ";
        cin>>ch;
    }

    return 0;
}

int enqueue(int queue[], int elem)
{
    if(rear == SIZE-1)
```

```
    {
        return -1;
    }
    else if(rear == -1)
    {
        front = rear = 0;
        queue[rear] = elem;
    }
    else
    {
        rear++;
        queue[rear] = elem;
    }
    return 0;
}

int dequeue(int queue[])
{
    int retn;
    if(front == -1)
    {
        return -1;
    }
    else
    {
        retn = queue[front];
        if(front == rear)
        {
            front = rear = -1;
        }
        else
        {
            front++;
        }
    }
    return retn;
}

void display(int queue[], int front, int rear)
{
    if(front == -1)
    {
        return;
    }
    for(int i=front; i<rear; i++)
    {
```

```
        cout<<queue[i]<<" <- ";
    }
    cout<<queue[rear]<< endl;
}
```

Kode 6-3. QueueArrayDequeue.cpp

- Jelaskan kode program di atas!

6.3.4 Percobaan 6-4: Queue dengan Linked List Enqueue

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;

struct node
{
    int info;
    node *next;
} *front, *newptr, *save, *ptr, *rear;

node *create_new_node(int);
void insert(node *);
void display(node *);

int main()
{
    front = rear = NULL;
    int inf;
    int count=0;
    char ch='y';

    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan data pada antrian= ";
        cin>>inf;
        newptr = create_new_node(inf);
        if(newptr == NULL)
        {
            cout<<"Tidak dapat membuat node..!!..Keluar program.." << endl;
            exit(1);
        }
        insert(newptr);
        cout<<"Antrian berhasil ditambahkan..." << endl;
        cout<<"Queue (Depan...ke...Belakang):" << endl;
        display(front);
        cout<<"Mau menambahkan antrian? (y/n) ";
        cin>>ch;
    }
```

```
    return 0;
}

node *create_new_node(int x)
{
    ptr = new node;
    ptr->info = x;
    ptr->next = NULL;
    return ptr;
}

void insert(node *n)
{
    if(front == NULL)
    {
        front = rear = n;
    }
    else
    {
        rear->next = n;
        rear = n;
    }
}

void display(node *n)
{
    while(n != NULL)
    {
        cout<<n->info<<" -> ";
        n = n->next;
    }
    cout<<"!!\n";
}
```

Kode 6-4. QueueLinkedListEnqueue.cpp

- Jelaskan kode program di atas!

6.3.5 Percobaan 6-5: Queue dengan Linked List Dequeue

Ketik kode program di bawah ini:

```
#include<iostream>
using namespace std;

struct node
{
    int info;
    node *next;
```

```
} *front, *newptr, *save, *ptr, *rear;

node *create_new_node(int);
void insert(node *);
void delete_node_queue();
void display(node *);

int main()
{
    front = rear = NULL;
    int inf;
    int count=0;
    char ch='y';

    while(ch=='y' || ch=='Y')
    {
        cout<<"Masukkan data pada antrian= ";
        cin>>inf;
        newptr = create_new_node(inf);
        if(newptr == NULL)
        {
            cout<<"Tidak dapat membuat node..!!..Keluar program.." << endl;
            exit(1);
        }
        insert(newptr);
        cout<<"Antrian berhasil ditambahkan..." << endl;
        cout<<"Queue (Depan...ke...Belakang):" << endl;
        display(front);
        cout<<"Mau menambahkan antrian? (y/n) ";
        cin>>ch;
    }

    do {
        cout<<"Queue (Depan...ke...Belakang):" << endl;
        display(front);
        if(count == 0)
        {
            cout<<"Mau mengeluarkan elemen dari antrian? (y/n) ";
            count++;
        }
        else
        {
            cout<<"Mau mengeluarkan elemen dari antrian lagi? (y/n) ";
        }
        cin>>ch;
        if(ch=='y' || ch=='Y')
```

```
        {
            delete_node_queue();
        }
        cout << endl;
    } while(ch=='y' || ch=='Y');

    return 0;
}

node *create_new_node(int x)
{
    ptr = new node;
    ptr->info = x;
    ptr->next = NULL;
    return ptr;
}

void insert(node *n)
{
    if(front == NULL)
    {
        front = rear = n;
    }
    else
    {
        rear->next = n;
        rear = n;
    }
}

void delete_node_queue()
{
    if(front == NULL)
    {
        cout<<"Overflow..!!..Keluar program.." << endl;
        exit(2);
    }
    else
    {
        ptr = front;
        front = front->next;
    }
}

void display(node *n)
{
```

```
    while(n != NULL)
    {
        cout<<n->info<<" -> ";
        n = n->next;
    }
    cout<<"!!\n";
}
```

Kode 6-5. QueueLinkedListDequeue.cpp

- Jelaskan kode program di atas!

6.4 TUGAS AKHIR

- Buat flowchart Queue dengan Array dan Queue dengan Linked List!
- Buat program antrian dokter yang datanya disimpan dalam Linked List!

DAFTAR PUSTAKA

C, A. R. (2012). *HANDOUT ALGORITMA PEMROGRAMAN DAN STRUKTUR DATA*. Yogyakarta: PRODI SISTEM INFORMASI UKDW.

CodesCracker. (2018). *C++ Tutorial*. Diambil kembali dari CodesCracker: https://codescracker.com/cpp/

cplusplus.com. (2017). *Data Structures - C++ Tutorials*. Diambil kembali dari C++ Tutorials: http://www.cplusplus.com/doc/tutorial/structures/

Mardiana. (2014). Materi Kuliah Struktur Data. Bandar Lampung, Lampung, Indonesia.

Programiz. (2018). *Learn C++*. Diambil kembali dari Programiz: https://www.programiz.com/cpp-programming

Tutorials Point. (2018). *C++ Tutorial*. Diambil kembali dari Tutorials Point: https://www.tutorialspoint.com/cplusplus/