ISBN : 978-623-7833-58-1

UNPAM PRESS
Lembaga Penerbit & Publikasi

STRUKTUR DATA

Achmad Udin Zailani | Budi Apriyanto | Hadi Zakaria

Program Studi Teknik Informatika S-1 | Universitas Pamulang

STRUKTUR DATA

Penyusun :
Achmad Udin Zailani
Budi Apriyanto
Hadi Zakaria

UNPAM PRESS
Lembaga Penerbit & Publikasi
Jl. Surya Kencana No. 1 Pamulang
Gd. A, Ruang 211 Universitas Pamulang
Tangerang Selatan - Banten

STRUKTUR DATA

Penulis :
Achmad Udin Zailani
Budi Apriyanto
Hadi Zakaria

ISBN : 978-623-7833-58-1

Editor :
Samsoni

Desain Sampul:
Ubaid Al Faruq, M.Pd.

Tata Letak:
Aden, S.Si., M.Pd.

Penerbit:
Unpam Press

Redaksi:
Jl. Surya Kencana No. 1
R. 212, Gd. A Universitas Pamulang Pamulang | Tangerang Selatan | Banten
Tlp/Fax: **021**. 741 2566 – 7470 9855 Ext: 1073
Email: unpampress@unpam.ac.id

Cetakan pertama, 2020

LEMBAR IDENTITAS ARSIP

Data Publikasi Unpam Press

| Lembaga Pengembangan Pendidikan dan Pembelajaran

Gedung A. R. 211 Kampus 1 Universitas Pamulang

Jalan Surya Kencana Nomor 1. Pamulang Barat, Tangerang Selatan, Banten.

Website: www.unpam.ac.id | **email**: unpampress@unpam.ac.id

Struktur Data / Achmad Udin Zailani, S.Kom.,M.Kom.. Budi Apriyanto, S.Kom., M.Kom., Hadi Zakaria., S.Kom., M.Kom., M.M

ISBN : 978-623-7833-58-1

Ketua Unpam Press : Pranoto
Koordinator Editorial dan Produksi: Ubaid Al Faruq, Ali Madinsyah
Koordinator Bidang Hak Cipta : Susanto
Koordinator Publikasi dan Dokumentasi : Aden
Desain Cover : Ubaid Al Faruq
:

Cetakan pertama, 2020

MATA KULIAH
STRUKTUR DATA

IDENTITAS MATAKULIAH

Program Studi	: Teknik Informatika
Mata Kuliah/Kode	: Struktur Data
Jumlah SKS	: 3 SKS
Prasyarat	:
Deskripsi Mata Kuliah	: Mata Kuliah ini mengajarkan materi lanjut dari pemrograman seperti pointer, struct, dsb. Selain itu juga beberapa struktur data yang digunakan dalam pemrograman, baik yang statis atau dinamis
Capaian Pembelajaran	: Mahasiswa mampu memecahkan masalah menjadi sebuah algoritma (langkah-langkah) yang akan dijalankan oleh komputer, kemudian mengimplementasikannya menjadi sebuah program computer sesuai kaidah ilmu pemrograman.
Penyusun	: Achmad Udin Zailani, S.Kom., M.Kom (Ketua) Budi Apriyanto, S.Kom., M.Kom (Anggota) Hadi Zakaria, S.Kom., M.Kom., M.M (Anggota)

Ketua Program Studi	Ketua Team Teaching
Dr. Ir Sewaka, M.M	Achmad Udin Zailani, S.Kom., M.Kom
NIDK. 8842760018	NIDN. 0429058303

KATA PENGANTAR

Dalam modul Struktur Data ini tidak dibahas teknik dasar penyusunan algoritma secara detail, tetapi lebih kearah memanfaatkan langsung teknik – teknik dasar yang berkaitan dengan pengelolan data yang terkait dengan struktur data seperti *stack, queue, linked list, sorting* dan *searching* dalam membangun suatu aplikasi.

Pada modul ini mahasiswa diperkenalkan dari bahasa pemograman C/C++ yang digunakan sebagai bahasa pengantar, selanjutnya dijelaskan apa itu tipe data yang ada di C++, Apa itu Array? Dan masuk kedalam data dinamis dan Statis. Setelah itu masuk ke pengelolaan data dengan memanfaatkan teknik *stack, queue, linked list* dan memberikan contoh aplikasi sederhana. Diharapkan dengan teknik seperti itu mahasiswa dapat menerapkan konsep teori dengan aplikasi yang bisa diterapkan dalam pengelolaan data yang lebih efiseien. Sehingga maksud tujuan dari modul ini mahasiswa mampu menerapkan konsep struktur data untuk data statis dan dinamis.

Rasa syukur tidak lupa penulis panjatkan kepada Allah SWT, atas limpahan rahmatNya, sehingga penulis dapat menyelesaikan modul ini. Mudah-mudahan modul ini dapat berguna sebagai bahan referensi untuk mahasiswa dan dosen pengampu mata kuliah struktur data.

Tangerang Selatan, 30 September 2020

Tim Penyusun

Achmad Udin Zailani, S.Kom., M.Kom

NIDN. 0429058303

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

PERTEMUAN 1
TIPE DATA & HIRARKI

A. TUJUAN PEMBELAJARAN

Tujuan dari bab ini adalah untuk membedakan dan menerapkan beragam jenis tipe data, hirarki data dan dapat memecahkan masalah sebanyak mungkin. Dan focus utama dipemahaman elemen dasar algoritma, pentingnya analisis algoritma, dan kemudian perlahan-lahan bergerak ke arah topik lain seperti yang disebutkan di atas. Setelah menyelesaikan bab ini, Anda harus dapat menemukan kompleksitas dari setiap algoritma yang diberikan (terutama fungsi rekursif):

1. Mampu membedakan dan menerapkan beragam jenis tipe data
2. Mampu membedakan dan menerapkan hirarki data

B. URAIAN MATERI

1. **Variabel**

Sebelum menuju ke definisi variabel, mari kita hubungkan dengan persamaan matematika. Kita semua mungkin telah memecahkan banyak persamaan matematika sejak kecil. Sebagai contoh, perhatikan persamaan di bawah ini:

$$\boldsymbol{X^2 + 2y - 2 = 1}$$

Kita tidak perlu khawatir tentang penggunaan persamaan ini. Dimana persamaan memiliki Variabel (x dan y), yang mempunyai nilai (data). Itu berarti variabel (x dan y) adalah *placeholder* untuk mewakili data. Demikian pula, dalam pemrograman ilmu komputer kita perlu sesuatu untuk mengikat data, dan variabel adalah cara untuk melakukan itu.

Pada pemrograman variabel merupakan lokasi atau tempat di dalam memori yang mampu menyimpan data sementara pada suatu program, data tersebut dapat diubah, disimpan atau ditampilkan kapanpun pada saat dibutuhkan. Maka dalam variabel harus memiliki nama, dimana nama variabel tidak boleh sama dengan variable lainnya. Setiap variabel memiliki alamat sendiri pada memori komputer, kita hanya perlu meyebutkan nama variabel dimana data di simpan, maka komputer akan mampu menemukan alamat variabel tersebut tersimpan pada memori komputer.

Ada beberapa syarat yang diperlukan dalam pemberian nama variabel yang ditentukan oleh pembuat program seperti berikut :

1. Penulisan nama variabel tidak boleh sama dengan nama keyword dan *function*.
2. Penulisan nama variabel maksimum 32 karakter.
3. Penulisan nama variabel harus diawali dengan huruf atau garis bawah (*underscore* _), karakter berikutnya boleh angka, huruf atau garis bawah
4. Penulisan nama variabel tidak boleh ada spasi.

2. **Tipe Data**

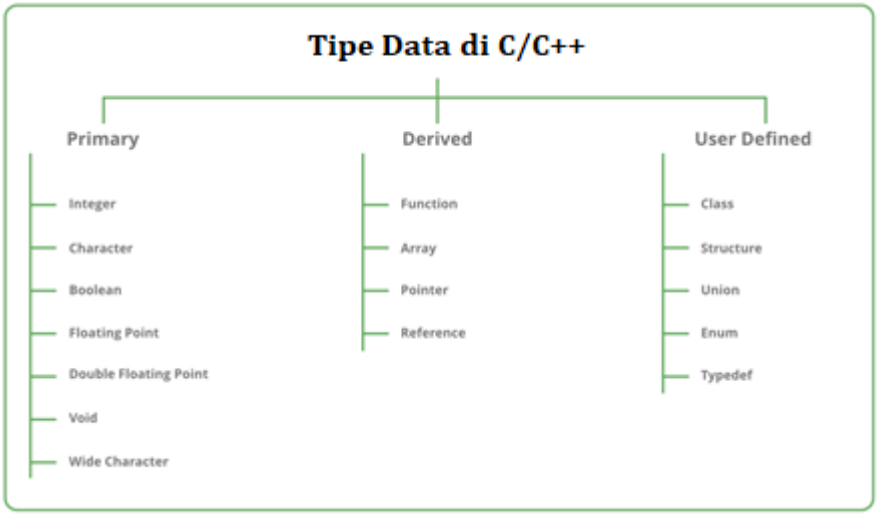
Dalam persamaan yang disebutkan di atas, variabel x dan y dapat mengambil nilai apa pun seperti bilangan integral (10, 20), bilangan riil (0,23, 5,5), atau hanya 0 dan 1. Untuk memecahkan persamaan, kita perlu menghubungkan mereka dengan jenis data yang dapat mereka ambil, dan jenis data adalah sesuatu yang digunakan dalam pemrograman ilmu komputer untuk tujuan ini. Jenis data dalam bahasa pemrograman adalah sekumpulan data dengan nilai yang telah ditetapkan. Contoh jenis data adalah: bilangan bulat, karakter, string, dll. Memori komputer semua diisi dengan 0 dan 1. Jika kita memiliki masalah dan kita ingin kode itu, sangat sulit untuk memberikan solusi dalam hal 0 dan 1. Untuk membantu pengguna, bahasa pemrograman dan kompiler memberikan jenis data. Sebagai contoh, *integer* mengambil 2 *byte* , *Float* mengambil 4 *byte*, dll. Ini menyatakan bahwa dalam memori kita menggabungkan 2 byte (16 bit) dan menyebutnya integer. Demikian pula, menggabungkan 4 byte (32 bit) dan menyebutnya Float. Jenis data mengurangi upaya pengkodean. Secara umum, ada dua jenis data:

- Data yang ditetapkan sistem
- Data yang ditetapkan pengguna

a. **Jenis data yang ditetapkan sistem (tipe data utama)**

Tipe data utama merupakan tipe data yang didefinisikan oleh sistem. Tipe data utama yang disediakan oleh banyak bahasa pemrograman yaitu: *int, Float, char, Double, Bool*, dll. Jumlah bit yang dialokasikan untuk setiap jenis data dasar tergantung pada bahasa pemrograman, kompilator dan sistem operasi. Untuk jenis data dasar yang sama, bahasa yang berbeda mungkin menggunakan ukuran yang berbeda. Tergantung pada ukuran jenis data, total nilai yang tersedia (domain) juga akan berubah. Sebagai contoh, "int" mungkin mengambil 2 byte atau 4 byte. Jika dibutuhkan 2 byte (16 bit), maka total nilai yang mungkin adalah minus 32.768

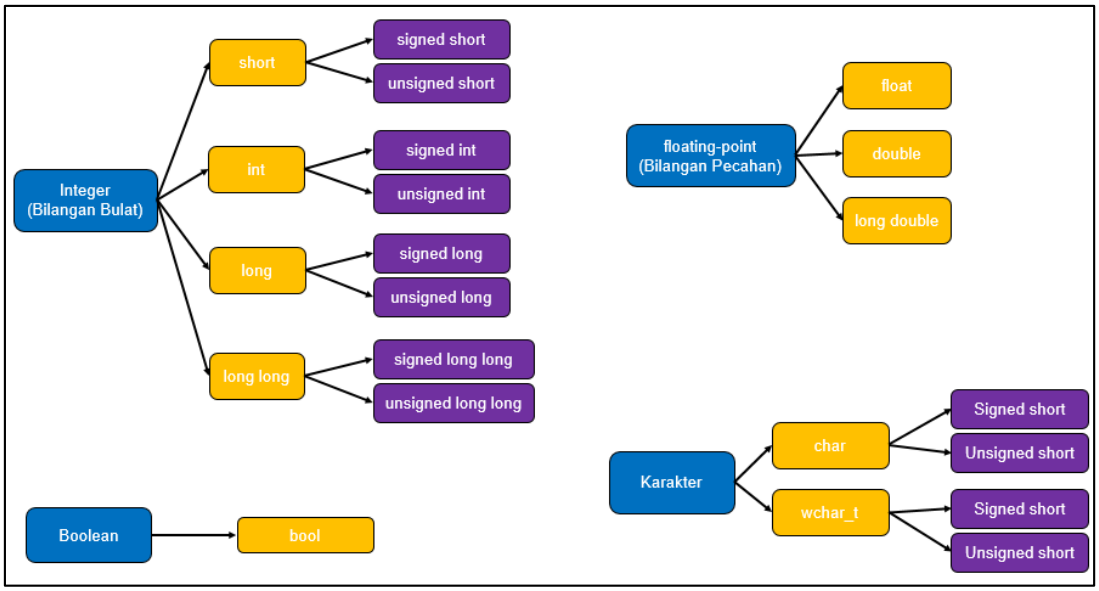
untuk Plus 32.767 (-215 ke 215-1). Jika dibutuhkan 4 byte (32 bit), maka nilai yang mungkin adalah antara-2.147.483.648 dan + 2.147.483.647 (-231 untuk 231-1). Hal yang sama terjadi dengan jenis data lainnya.



Gambar 1.1 Jenis Tipe data di C/C++

Jenis data di C++ terutama dibagi menjadi tiga jenis:

1) Tipe Data Utama: Merupakan data standar yang bisa digunakan langsung oleh pengguna untuk mendeklarasikan suatu variabel. Misal: *int, char, float, bool* dst. Adapun data yang biasa digunakan di C++ yaitu:
    - *Integer*
    - *Character*
    - *Boolean*
    - *Floating Point*
    - *Double Floating Point*
    - *void*
    - *Wide character*



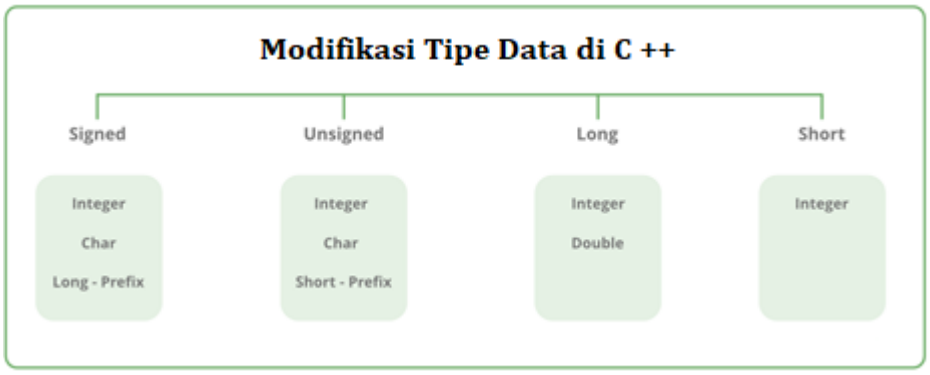
Gambar 1.2 Tipe Data Utama

2) Tipe Data Turunan adalah tipe data yang berasal dari tipe data utama. Tipe data ini dibagi menjadi empat jenis yaitu:
   - Fungsi
   - *Array*
   - *Pointer*
   - *Referensi*
3) Tipe Data Abstrak atau Tipe Data Yang Ditentukan oleh Pengguna:. Contoh, Pengguna mendefinisikan kelas di C++ atau struktur. Biasanya C++ menyediakan tipe data ini yaitu:
   - Kelas
   - Struktur
   - Union
   - Enumerasi
   - Typedef

Berikut data dasar yang tersedia di C++.

- Bilangan bulat: Kata kunci yang digunakan untuk jenis data bilangan bulat int. Bilangan bulat biasanya memerlukan 4 byte ruang memori dan berkisar dari -2147483648 hingga 2147483647.

- Karakter merupakan jenis tipe data yang digunakan untuk menyimpan karakter. Adapun kata kunci yang digunakan untuk jenis tipe data karakter yaitu char. Karakter biasanya membutuhkan 1 byte ruang memori dan berkisar dari -128 ke 127 atau 0 hingga 255.
- *Boolean* merupakan jenis data *boolean* digunakan untuk menyimpan nilai boolean atau logis. Variabel boolean dapat menyimpan nilai *true* atau *false*. Dimana kata kunci pada jenis data boolean yaitu bool.
- *Floating Point* merupakan tipe data floating point digunakan untuk menyimpan nilai floating point presisi tunggal atau nilai desimal. Kata kunci pada jenis data *floating point* yatu *float*. Variabel float biasanya membutuhkan 4 byte ruang memori.
- Double Floating Point: Tipe data Floating Point Ganda digunakan untuk menyimpan nilai floating point presisi ganda atau nilai desimal. Kata kunci yang digunakan untuk tipe data floating point ganda adalah dua kali lipat. Variabel ganda biasanya memerlukan 8 byte ruang memori.
- void: Void berarti tanpa nilai apapun. void datatype mewakili entitas yang tidak berharga. Jenis data void digunakan untuk fungsi tersebut yang tidak mengembalikan nilai.
- Wide character: Tipe data karakter lebar juga merupakan tipe data karakter tetapi jenis data ini memiliki ukuran yang lebih besar daripada tipe data 8-bit normal. Diwakili oleh wchar_t. Hal ini umumnya 2 atau 4 byte panjang.



Gambar 1.2 Modifikasi Tipe Data di C++

Dalam C++ disediakan tipe data yang dapat dimodifikasi yaitu: *Signed, Unsigned* dan *Short Long.*

Tabel di bawah ini meringkas tipe data turunan yang dimodifikasi dengan jangkauan nilai numerik dan ukuran penyimpanannya:

Tabel 1.1 Tipe data turunan dan ukurannya

No	Sebutan Tipe data	Penulisan dalam bahasa C/C++	Jumlah byte	Jangkauan nilai numerik
1.	Character	signed char	1	-128 s.d 127
		unsigned char	1	0 s.d 255
2.	Integer	short int	2	-32,768 s.d 32,767
		unsigned short int	2	0 s.d 65,535
		unsigned int	4	0 s.d 4,294,967,295
		Int	4	-2,147,483,648 s.d 2,147,483,647
		long int	8	-2,147,483,648 s.d 2,147,483,647
		unsigned long int	4	0 s.d 4,294,967,295
		long long int	8	-(2^63) s.d (2^63)-1
		unsigned long long int	8	0 s.d 18,446,744,073,709,551,615
3.	Floating point single precision	Float	4	
4.	Floating point double precision	Double	8	
		long double	12	
5.	Wide character	wchar_t	2 atau 4	1 wide character

Listing. Program C++ Tipe data dan ukurannya

```
// Program C++ Tipe data dan ukurannya
#include<iostream>
using namespace std;

int main()
{
   cout << "Ukuran dari char : " << sizeof(char)
     << " byte" << endl;
   cout << "Ukuran dari int : " << sizeof(int)
     << " bytes" << endl;
   cout << "Ukuran dari short int : " << sizeof(short int)
     << " bytes" << endl;
   cout << "Ukuran dari long int : " << sizeof(long int)
       << " bytes" << endl;
   cout << "Ukuran dari signed long int : " << sizeof(signed long int)
       << " bytes" << endl;
   cout << "Ukuran dari unsigned long int : " << sizeof(unsigned long int)
       << " bytes" << endl;
   cout << "Ukuran dari float : " << sizeof(float)
       << " bytes" <<endl;
   cout << "Ukuran dari double : " << sizeof(double)
       << " bytes" << endl;
   cout << "Ukuran dari wchar_t : " << sizeof(wchar_t)
       << " bytes" <<endl;

   return 0;
}
```

Output dari program diatas:

```
Ukuran dari char : 1 byte
Ukuran dari int : 4 bytes
Ukuran dari short int : 2 bytes
Ukuran dari long int : 8 bytes
Ukuran dari signed long int : 8 bytes
Ukuran dari unsigned long int : 8 bytes
Ukuran dari float : 4 bytes
Ukuran dari double : 8 bytes
Ukuran dari wchar_t : 4 bytes
```

Berdasarkan diskusi di atas, setelah kita memiliki data dalam variabel, kita memerlukan mekanisme untuk memanipulasi data tersebut untuk memecahkan masalah. Struktur data merupakan suatu cara tertentu untuk mengatur dan menyimpan data di dalam komputer untuk memproses data lebih efisien. Struktur data mempunyai format khusus dalam mengatur dan menyimpan data. Jenis struktur data umum meliputi array, file, liked list, *stack, queue, trees, graph* dan sebagainya. Tergantung pada organisasi elemen, struktur data diklasifikasikan ke dalam dua jenis:

1) Struktur data *linea*r: elemen diakses dalam urutan berurutan tetapi tidak wajib untuk menyimpan semua elemen secara berurutan. Contoh: Linked list, Stack, dan Queue.
2) Struktur data *non-linear*: elemen struktur data ini disimpan/diakses dalam urutan non-linear. Contoh: *Trees dan Graph*.

Sebelum mendefinisikan jenis data abstrak, perlu kita pertimbangkan tampilan yang berbeda dari jenis data yang ditentukan sistem. Kita semua tahu bahwa, secara default, semua tipe data dasar (int, Float, dll.) mendukung operasi dasar seperti penambahan dan pengurangan. Sistem ini menyediakan implementasi untuk jenis data utama. Untuk jenis data yang ditetapkan pengguna kita juga perlu mendefinisikan operasi. Implementasi untuk operasi ini dapat dilakukan saat kita ingin menggunakannya. Itu berarti, secara umum, jenis data yang didefinisikan pengguna didefinisikan bersama dengan operasinya.

Perbedan Struktur Data Linier dengan Struktur Data non-linier dapat dilihat pada tabel dibawah ini :

Tabel 1.2 Perbedaan Struktur Data Linier dan Non Linier

Kategori	**Struktur Data Linier**	**Struktur data *non-linear***
Definisi	Elemen diakses dalam urutan berurutan tetapi	Elemen struktur data ini disimpan/diakses dalam urutan non-linear
Urutan Data	Urutan linier	Tidak urutan linier
Pembacaan Element Data	Sangat mungkin dalam sekali jalan	Sangat sulit dalam sekali jalan
Implementasi	Mudah	Sulit

Contoh	Linked list, Stack, dan Queue	Trees dan Graph.

C. SOAL LATIHAN/TUGAS

Latihan.1

Kerjakan soal dibawah ini dengan menggunakan Bahasa pemograman C++ dan mengujinya di compiler C++

1. Diberikan tiga nilai dari tipe data yang berbeda. Anda diminta untuk menyimpan nilai-nilai ini dalam variabel tipe data yang sesuai dan mencetaknya.

   Input:
   Satu-satunya baris masukan berisi tiga nilai yang dipisahkan spasi dari jenis yang berbeda.
   Deskripsi tipe data dari nilai:
   1) Nilai pertama adalah tipe integer.
   2) Nilai kedua adalah tipe karakter.
   3) Nilai ketiga adalah tipe float.

   Output:
   Output tiga baris seperti yang dijelaskan:
   1) Baris pertama akan berisi nilai pertama.
   2) Baris kedua akan berisi nilai kedua
   3) Baris ketiga akan berisi nilai ketiga.

2. Tulis program yang meminta pengguna untuk mengetik lebar dan tinggi persegi panjang dan kemudian menampilkan area dan keliling persegi panjang tersebut ke layar.

3. Tulis program yang menanyakan nilai siswa. Skor adalah angka dari 0-100. Terjemahkan skor menjadi nilai sesuai dengan batas berikutnya:

skor> = 90 ==> "A"
skor> = 80 ==> "B"
skor> = 70 ==> "C"
skor> = 60 ==> "D"
apa pun ==> "F"

jika skor 100 cetak "Skor sempurna!"

4. Tulis program yang meminta pengguna mengetikkan koordinat 2 titik, A dan B (dalam bidang), lalu tulis jarak antara A dan B.

5. Tulis program yang meminta pengguna untuk mengetikkan 2 bilangan bulat A dan B dan menukar nilai A dan B

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Sjukani, M. (2012). *Struktur Data (Algoritma dan Struktur Data 2), Edisi 5.* Jakarta: Mitra Wacana Media.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 2
ARRAY DAN POINTER

A. TUJUAN PEMBELAJARAN

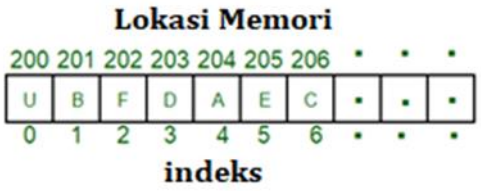
Pada bab ini menjelaskan implementasi Array 1, 2 dan banyak dimensi dan tipe data pointer yang digunakan dalam Bahasa pemrograman:

1. Mampu menerapkan Array 1, 2 dan banyak dimensi
2. Mampu menerapkan tipe data pointer yang digunakan dalam Bahasa pemrograman

B. URAIAN MATERI

Array adalah kumpulan item yang disimpan di lokasi memori yang berdekatan. Awalnya array digunakan untuk menyimpan beberapa item dengan jenis yang sama. Ini mempermudah penghitungan posisi setiap elemen dengan hanya menambahkan offset ke nilai dasar, yaitu lokasi memori elemen pertama dari array (biasanya dilambangkan dengan nama array).

Untuk kesederhanaan, kita dapat membayangkan sebuah deret armada tangga di mana pada setiap anak tangga diberi nilai (misalkan salah satu teman Anda). Di sini, Anda dapat mengidentifikasi lokasi teman Anda hanya dengan mengetahui hitungan langkah mereka. Ingat: “Lokasi indeks berikutnya tergantung pada tipe data yang kita gunakan”.

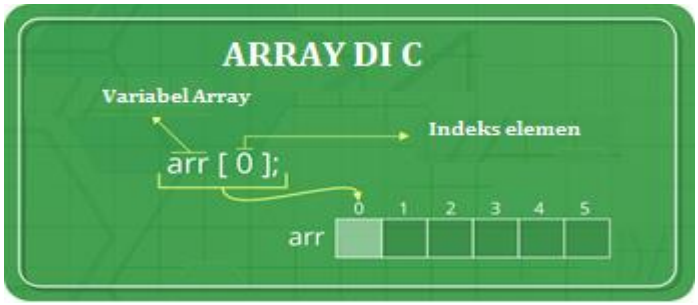


Gambar 2.1 Lokasi Memori

Gambar di atas dapat dilihat sebagai tampilan tingkat atas dari tangga tempat Anda berada di dasar tangga. Setiap elemen dapat diidentifikasi secara unik dengan indeksnya dalam larik (dengan cara yang sama seperti Anda dapat mengidentifikasi teman-teman Anda melalui langkah mereka pada contoh di atas).

Jenis pengindeksan dalam array:

- 0 (pengindeksan berbasis nol): Elemen pertama dari array diindeks oleh subskrip 0
- 1 (pengindeksan berbasis satu): Elemen kedua dari array diindeks oleh subskrip 1
- n (pengindeksan berbasis n): Indeks dasar array dapat dipilih secara bebas. Biasanya bahasa pemrograman yang memungkinkan pengindeksan berbasis n juga memungkinkan nilai indeks negatif dan tipe data skalar lainnya seperti enumerasi, atau karakter dapat digunakan sebagai indeks array.



Gambar 2.2 Array di C

Keuntungan menggunakan array:

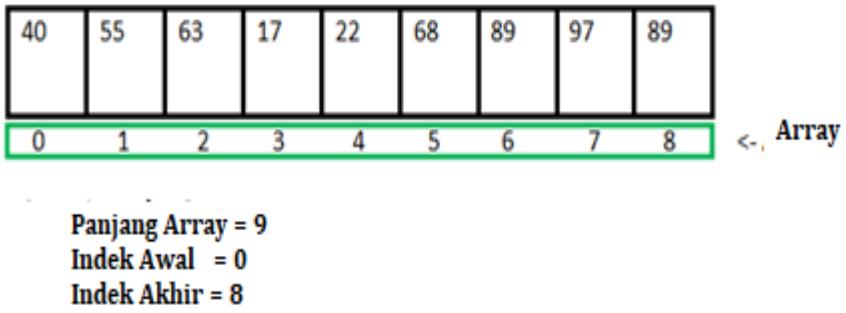
- Array memungkinkan akses elemen secara acak. Ini membuat pengaksesan elemen berdasarkan posisi lebih cepat.
- Array memiliki lokalitas cache yang lebih baik yang dapat membuat perbedaan kinerja yang cukup besar.

```
// Array karakter di C / C ++
 char arr1 [] = {'u', 'n', 'p', 'a', 'm'};

 // Array Integer di C / C ++
 int arr2 [] = {1, 2, 3, 4, 5};

 // Item pada indeks ke-i dalam array biasanya diakses
 // sebagai "arr [i]".  Misalnya arr1 [0] memberi kita 'u'
 // dan arr2 [3] memberi kita 4
```

Biasanya, array karakter disebut 'string', sedangkan array int atau float disebut dengan array.

1. **Array Dalam C / C ++**

Array di C atau C ++ merupakan sekumpulan item yang dapat disimpan pada lokasi memori . Yang mana yang berdekatan dan elemen dapat diakses secara acak menggunakan indeks array. Mereka digunakan untuk menyimpan jenis elemen yang sama karena dalam tipe data harus sama untuk semua elemen. Mereka dapat digunakan untuk menyimpan kumpulan tipe data dasar seperti *int, float, double, char,* dll dari tipe tertentu. Untuk menambahkannya, sebuah array dalam C atau C ++ dapat menyimpan tipe data turunan seperti *struktur, pointer*, dll. Diberikan di bawah ini adalah representasi indah dari sebuah array.

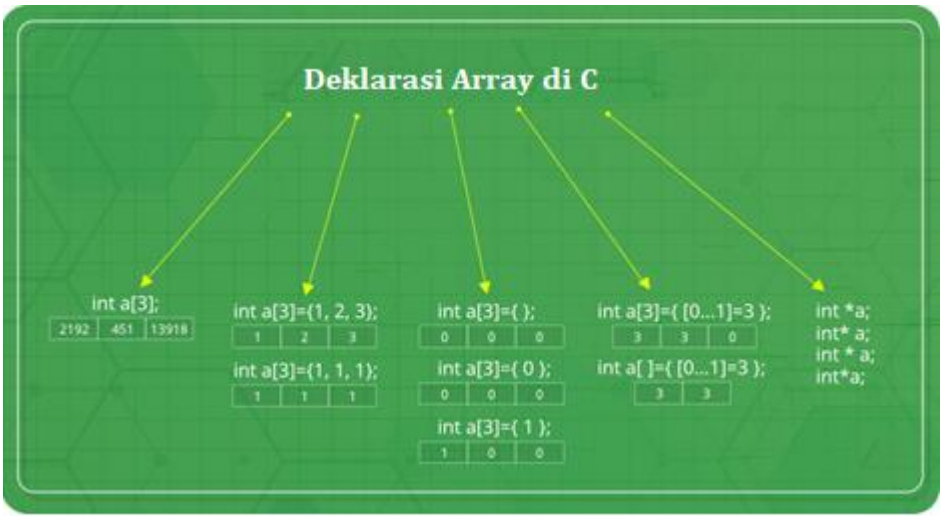


Gambar 2.3 Representasi Array

Mengapa kita membutuhkan array?

Kita dapat menggunakan variabel normal (v1, v2, v3, ..) ketika kita memiliki sejumlah kecil objek, tetapi jika kita ingin menyimpan instance dalam jumlah besar, akan sulit untuk mengelolanya dengan variabel normal. Ide dari sebuah array adalah untuk merepresentasikan banyak contoh dalam satu variabel.

Deklarasi array di C / C ++:



Gambar 2.4 Deklarasi array di C / C ++

Ada berbagai cara untuk mendeklarasikan sebuah array. Ini dapat dilakukan dengan menentukan jenis dan ukurannya, dengan menginisialisasi atau keduanya.

1) **Deklarasi array dengan menentukan ukuran**

```
// Deklarasi array dengan menentukan ukuran
int arr1 [10];

// Dengan versi C / C ++ mendeklarasikan array dengan ukuran yang
ditentukan pengguna
int n = 10;
int arr2 [n];
```

2) **Deklarasi array dengan menginisialisasi elemen**

```
// Deklarasi array dengan menginisialisasi elemen
int arr [ ] = {20, 30, 40, 50}

// Compiler  membuat array berukuran 4.
// di atas sama dengan "int arr [4] = {20, 30, 40, 50}"
```

3) **Deklarasi array dengan menentukan ukuran dan menginisialisasi elemen**

```
// Deklarasi array dengan menentukan ukuran dan menginisialisasi elemen
int arr [6] = {20, 30, 40, 50}

// Compiler membuat array berukuran 6, menginisialisasi terlebih dahulu 4 elemen seperti yang ditentukan oleh pengguna dan dua elemen lainnya adalah 0.
// di atas sama dengan "int arr [] = {20, 30, 40, 50, 0, 0}"
```

Keuntungan dari Array di C / C ++:

1. Akses acak elemen menggunakan indeks array.
2. Gunakan lebih sedikit baris kode karena ini membuat satu array dari banyak elemen.
3. Akses mudah ke semua elemen.
4. Traversal melalui array menjadi mudah menggunakan satu loop.
5. Penyortiran menjadi mudah karena dapat dilakukan dengan menulis lebih sedikit baris kode.

Kekurangan Array di C / C ++:

1. Memungkinkan sejumlah elemen untuk dimasukkan yang ditentukan pada saat deklarasi. Tidak seperti linked list, array di C tidak dinamis.
2. Penyisipan dan penghapusan elemen dapat memakan biaya karena elemen tersebut perlu dikelola sesuai dengan alokasi memori baru.

2. **Array Multidimensi Dalam C / C ++**

Dalam C / C ++, kita dapat mendefinisikan array multidimensi dengan kata sederhana sebagai array dari array. Data dalam array multidimensi disimpan dalam bentuk tabel (dalam urutan mayor baris).

Bentuk umum untuk mendeklarasikan array berdimensi-N:

Tipe_data nama_array [size1] [size2] .... [sizeN];

Tipe_data : Jenis data yang akan disimpan dalam array.

Di sini tipe_data adalah tipe data C / C ++ yang valid

nama_array : Nama array

size1, size2, ..., sizeN : Ukuran dimensi

Contoh :

Array dua dimensi:

int 2_D [20] [30];

Array tiga dimensi:

int 3_D [20] [30] [50];

Ukuran array multidimensi

Jumlah total elemen yang dapat disimpan dalam array multidimensi dapat dihitung dengan mengalikan ukuran semua dimensi.

Sebagai contoh:

Array **int x [20] [30]** dapat menyimpan total (20 * 30) = 600 elemen.

Maka untuk Array **int x [10] [20] [30]** mampu menyimpan total (10 * 20 * 30) = 6000 elemen.

3. **Array Dua Dimensi**

Array dua dimensi merupakan bentuk paling sederhana dari array multidimensi. Array dua dimensi bisa kita lihat sebagai array satu dimensi agar lebih mudah dipahami.

Bentuk dasar mendeklarasikan array dua dimensi dengan ukuran x, y:

Sintak:

tipe_data nama_array [x] [y];

tipe_data : Jenis data yang akan disimpan. Tipe data C / C ++ yang valid.

Kita dapat mendeklarasikan array integer dua dimensi mengatakan 'x' dengan ukuran 20,30 sebagai:

```
int x [20] [30];
```

Elemen dalam array dua dimensi biasanya disebut x [i] [j] di mana i yaitu nomor baris dan 'j' yaitu nomor kolom.

Array dua dimensi dapat dilihat sebagai tabel dengan baris 'x' dan 'y' di mana nomor baris berkisar dari 0 sampai (x-1) dan nomor kolom berkisar dari 0 sampai (y-1). Array dua dimensi 'x' dengan 3 baris dan 3 kolom ditampilkan di bawah ini:

	Kolom 0	Kolom 1	Kolom 2
Baris 0	x[0][0]	x[0][1]	x[0][2]
Baris 1	x[1][0]	x[1][1]	x[1][2]
Baris 2	x[2][0]	x[2][1]	x[2][2]

Gambar 2.5 Deklarasi Array 2 dimensi

Menginisialisasi Array Dua Dimensi : Ada dua cara untuk menginisialisasi array Dua Dimensi.

Metode Pertama :

```
int x [4] [5] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19}
```

Array di atas memiliki 4 baris dan kolom. Elemen-elemen dalam kurung kurawal dari kiri ke kanan juga disimpan dalam tabel dari kiri ke kanan. Elemen akan diisi dalam array secara berurutan, 4 elemen pertama dari kiri pada baris pertama, 4 elemen berikutnya pada baris kedua, dan seterusnya.

Metode yang Lebih Baik :

```
int x [4] [5] = {{0,1,2,3,4}, {5,6,7,8,9}, {10,11,12,13,14},{15,16,17,18,19}};
```

Jenis inisialisasi ini menggunakan tanda kurung bersarang. Setiap set kawat gigi bagian dalam mewakili satu baris. Dalam contoh di atas, total ada tiga baris jadi ada tiga set kawat gigi bagian dalam.

Mengakses Elemen Array Dua Dimensi: Elemen dalam array Dua Dimensi diakses menggunakan indeks baris dan indeks kolom. Contoh:

```
int x [2] [1];
```

Contoh di atas mewakili elemen yang ada di baris ketiga dan kolom kedua.

Catatan : Dalam array jika ukuran array adalah N. Indeksnya akan dari 0 hingga N-1. Oleh karena itu, untuk indeks baris 2 nomor baris adalah 2 + 1 = 3.

Untuk mengeluarkan semua elemen dari array Dua Dimensi kita dapat menggunakan *nested for loops*. Kami akan membutuhkan dua untuk loop. Satu untuk melintasi baris dan satu lagi untuk melintasi kolom.

Listing Program C++ untuk mencetak Elemen a

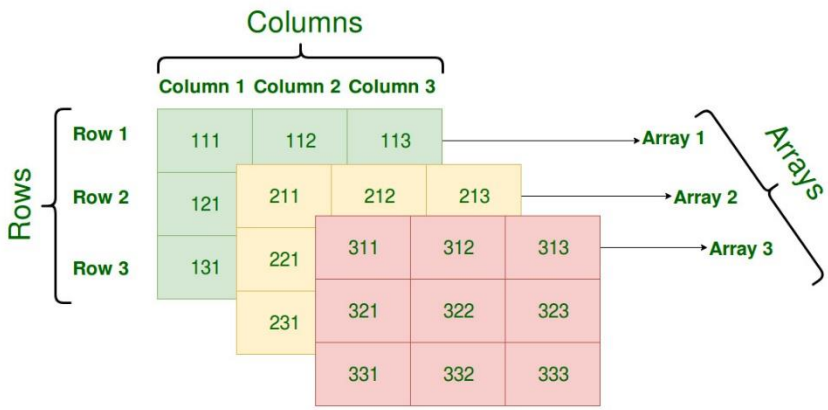
```
// Program C++ untuk mencetak elemen a
// Array dua dimensi
#include<iostream>
using namespace std;
int main()
{
    // array dengan 4 baris dan 3 kolom.
    int x[4][3] = {{0,1,2}, {3,4,5}, {6,7,8},{9,10,11}};
    // // keluarkan nilai setiap elemen array
    for ( int i = 0; i < 4; i++)
    {
        for ( int j = 0; j < 3; j++)
        {
```

```
            cout << "Elemen di x[" << i
                << "][" << j << "]: " ;
            cout << x[i][j]<<endl;
        }
    }
    return 0;
}
```

Output dari program diatas:

```
Elemen di x[0][0]: 0
Elemen di x[0][1]: 1
Elemen di x[0][2]: 2
Elemen di x[1][0]: 3
Elemen di x[1][1]: 4
Elemen di x[1][2]: 5
Elemen di x[2][0]: 6
Elemen di x[2][1]: 7
Elemen di x[2][2]: 8
Elemen di x[3][0]: 9
Elemen di x[3][1]: 10
Elemen di x[3][2]: 11
```

4. **Array Tiga Dimensi**



Gambar 2.6 Deklarasi Array 3 Dimensi

Inisialisasi Array Tiga Dimensi : Inisialisasi dalam array Tiga Dimensi sama dengan array Dua dimensi. Perbedaannya adalah dengan bertambahnya jumlah dimensi maka jumlah kurung kurawal juga akan bertambah.

Metode 1 :

```
int x [2] [3] [5] =
                    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
                    22,23,24,25,26,27,28,29};
```

Metode yang Lebih Baik :

```
int x [2] [3] [5] =
                    {{{0,1,2,3,4}, {5,6,7,8,9}, {10,11,12,13,14}},
                    {{15,16,17,18,19},                    {20,21,22,23,24},
                  {25,26,27,28,29}}};
```

Mengakses elemen dalam Array Tiga Dimensi : Mengakses elemen dalam Array Tiga Dimensi juga mirip dengan yang ada di Array Dua Dimensi. Perbedaannya adalah kita harus menggunakan tiga loop, bukan dua loop untuk satu dimensi tambahan dalam Array Tiga Dimensi.

Dengan cara serupa, kita dapat membuat array dengan sejumlah dimensi. Namun kompleksitas juga meningkat dengan bertambahnya jumlah dimensi. Array multidimensi yang paling banyak digunakan adalah Array Dua Dimensi

Listing Program C ++ untuk mencetak elemen pada Array Tiga Dimensi

```
// Program C ++ untuk mencetak elemen pada Array Tiga Dimensi
#include<iostream>
using namespace std;
int main()
{
    // menginisialisasi array 3 dimensi
    int x[3][2][3] =
    {
```

```
        { {0,1,2}, {3,4,5} },
        { {6,7,8}, {9,10,11} },
        { {12,13,14},{15,16,17}}
    };
    // keluarkan nilai setiap elemen
    for ( int i = 0; i < 3; ++i)
    {
        for ( int j = 0; j < 2; ++j)
        {
            for ( int k = 0; k < 3; ++k)
            {
                cout << "Elemen di x[" << i << "][" << j
                    << "][" << k << "] = " << x[i][j][k]
                    << endl;
            }
        }
    }
    return 0;
}
```

Output dari program diatas:

```
Elemen di x[0][0][0] = 0
Elemen di x[0][0][1] = 1
Elemen di x[0][0][2] = 2
Elemen di x[0][1][0] = 3
Elemen di x[0][1][1] = 4
Elemen di x[0][1][2] = 5
Elemen di x[1][0][0] = 6
Elemen di x[1][0][1] = 7
Elemen di x[1][0][2] = 8
Elemen di x[1][1][0] = 9
Elemen di x[1][1][1] = 10
Elemen di x[1][1][2] = 11
Elemen di x[2][0][0] = 12
Elemen di x[2][0][1] = 13
```

```
Elemen di x[2][0][2] = 14
Elemen di x[2][1][0] = 15
Elemen di x[2][1][1] = 16
Elemen di x[2][1][2] = 17
```

5. **Array vs Pointer**

Array dan pointer adalah dua hal yang berbeda (kita dapat memeriksa dengan menerapkan sizeof). Kebingungan terjadi karena nama array menunjukkan alamat elemen pertama dan array selalu dilewatkan sebagai pointer (bahkan jika kita menggunakan tanda kurung siku).

Apa itu vektor di C ++?

Vektor di C ++ adalah kelas di STL yang merepresentasikan sebuah array. Keuntungan vektor dibandingkan array normal adalah,

- Kita tidak membutuhkan pass size sebagai parameter tambahan ketika kita mendeklarasikan vektor, misalnya, Vektor mendukung ukuran dinamis (kita tidak harus menentukan ukuran vektor pada awalnya). Kami juga dapat mengubah ukuran vektor.
- Vektor memiliki banyak fungsi bawaan seperti, menghilangkan elemen, dll.

C. SOAL LATIHAN/TUGAS

Latihan 2.

Kerjakan soal dibawah ini dengan teliti.

1. Dideklarasikan Array 1 Dimensi yang dibuat dengan char x[12]. Jika diketahui &x[0] = 1000 Hexadecimal. Ditanya alamat elemen x[8] atau &x[8] = …..?
2. Dideklarasikan Array 1 Dimensi yang dibuat dengan int x[15]. Jika diketahui &x[3] = 1000 H, Ditanya &x[9] =…?
3. Dideklarasikan Array 2 Dimensi yang dibuat dengan float x[5][8]. Jika diketahui &x[0] [0]= 1000 H, Ditanya &x[2] [4]=…?
4. Dideklarasikan Array 2 Dimensi yang dibuat dengan long x[12][14]. Jika diketahui &x[0] [0]= 1000 H, Ditanya &x[2] [4]=…?

5. Dideklarasikan Array 3 Dimensi yang dibuat dengan int x [2][3] [5]. Jika diketahui &x [1][1] [4]= 12EF H, Ditanya &x [0][0] [3]= ….?

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Sjukani, M. (2012). *Struktur Data (Algoritma dan Struktur Data 2), Edisi 5.* Jakarta: Mitra Wacana Media.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 3
STRUKTUR / RECORD

A. TUJUAN PEMAHAMAN PEMBELAJARAN

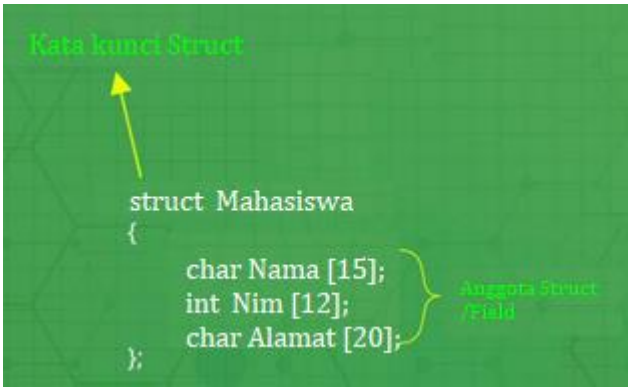
Pada bab ini akan diharapkan mampu memahami dan menerapkan:

1. Mampu memahami dan menerapkan struktur, array dalam struktur dan struktur dalam struktur

B. URAIAN MATERI

Kita sering menemukan situasi di mana kita perlu menyimpan sekelompok data baik jenis data serupa atau jenis data yang tidak serupa. Kita telah melihat Array di C ++ yang digunakan untuk menyimpan set data dari jenis data serupa di lokasi memori yang bersebelahan.

Tidak seperti Array, Struktur dalam C++ adalah jenis data yang ditentukan pengguna yang digunakan untuk menyimpan sekelompok item jenis data yang tidak serupa.



Gambar 3.1 Struktur di C++

Bagaimana cara membuat struktur?

Kata kunci 'struct' digunakan untuk membuat struktur. Sintaks umum untuk membuat struktur seperti yang ditunjukkan di bawah ini:

```
struct structureName{
   member1;
   member2;
   member3;
   .
   .
   .
   memberN;
};
```

Struktur di C++ dapat berisi dua jenis anggota:

1. Anggota Data: Anggota ini adalah variabel C++ normal. Kita dapat membuat struktur dengan variabel dari jenis data yang berbeda di C ++.
2. Fungsi Anggota: Anggota ini adalah fungsi C++ normal. Seiring dengan variabel, kita juga dapat menyertakan fungsi di dalam deklarasi struktur.

Bagaimana cara membuat struktur?

Pada pembuatan database biasanya data terdiri dari: *field, record,* dan *file*. *Field* sering disebut satuan terkecil dari data dan sedangkan *record* merupakan kumpulan dari *field* sementara *file* terdiri dari kumpulan *record* .

Struktur adalah kumpulan variabel dari tipe data yang berbeda di bawah satu nama. Ini mirip dengan kelas dalam hal itu, keduanya menyimpan kumpulan data dari tipe data yang berbeda.

Misalnya: Anda ingin menyimpan beberapa informasi tentang Seseorang yang terdiri dari informasi : nama, usia dan gajinya. Anda dapat dengan mudah membuat nama variabel yang berbeda, nama, usia dan gaji untuk menyimpan informasi ini secara terpisah.

Namun, di masa mendatang, Anda ingin menyimpan informasi tentang banyak orang. Sekarang, Anda perlu membuat variabel berbeda untuk setiap informasi per orang: nama1, usia1, gaji1, nama2, usia2, gaji2. Anda dapat dengan mudah memvisualisasikan seberapa besar dan dan banyak data yang akan terbentuk.

Pendekatan yang lebih baik adalah dengan mengumpulkan semua informasi terkait di bawah satu Nama Pribadi, dan menggunakannya untuk setiap orang. Sekarang, kodenya juga terlihat jauh lebih bersih, mudah dibaca, dan efisien. Kumpulan semua informasi terkait di bawah satu nama Pribadi adalah sebuah struktur.

Bagaimana cara mendeklarasikan struktur dalam pemrograman C ++?

Kata kunci struct mendefinisikan tipe struktur diikuti dengan pengenal (nama struktur). Kemudian di dalam kurung kurawal, Anda bisa mendeklarasikan satu atau lebih anggota (mendeklarasikan variabel di dalam kurung kurawal) dari struktur itu. Sebagai contoh:

```
struct Pribadi
{
    char nama [50];
    int usia;
    float gaji;
};
```

Di sini didefinisikan struktur pribadi yang memiliki tiga anggota: nama, usia dan gaji. Saat struktur dibuat, tidak ada memori yang dialokasikan. Definisi struktur hanyalah cetak biru untuk pembuatan variabel. Anda bisa membayangkannya sebagai tipe data. Saat Anda mendefinisikan integer seperti di bawah ini:

```
int KTP
```

Int menentukan bahwa, variabel KTP hanya dapat menampung elemen integer. Demikian pula, definisi struktur hanya menetapkan itu, properti apa yang dimiliki variabel struktur ketika ia didefinisikan.

Catatan: Ingatlah untuk mengakhiri deklarasi dengan titik koma (;)

Bagaimana cara mendefinisikan variabel struktur?

Setelah Anda mendeklarasikan struktur Pribadi seperti di atas. Anda dapat mendefinisikan variabel struktur sebagai:

```
Pribadi bill;
```

Di sini, didefinisikan bill struktur variabel yang merupakan tipe struktur Pribadi. Ketika variabel struktur ditentukan, hanya memori yang dibutuhkan dialokasikan oleh kompilator. Mengingat Anda memiliki sistem 32-bit atau 64-bit, memori float adalah 4 byte, memori int adalah 4 byte dan memori char adalah 1 byte. Oleh karena itu, 58 byte memori dialokasikan untuk bill struktur variabel.

Bagaimana cara mengakses anggota struktur? Anggota variabel struktur diakses menggunakan operator titik (.).

Misalkan, Anda ingin mengakses usia bill struktur variabel dan menetapkannya 50 untuk itu. Anda dapat melakukan tugas ini dengan menggunakan kode berikut ini:

```
bill.usia = 50;
```

Contoh 1.

Listing Program Program C ++ untuk menetapkan data anggota struktur variabel dan menampilkannya

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

struct Pribadi
{
   char Nama[50];
   int Usia;
   float Gaji;
};

int main()
{
   Pribadi p1;

   cout << "Masukan Nama Lengkap: ";
   cin.get(p1.Nama, 50);
   cout << "Masukkan Usia: ";
   cin >> p1.Usia;
   cout << "Masukkan Gaji: ";
   cin >> p1.Gaji;

   cout << "\nMenampilkan Informasi." << endl;
   cout << "Nama: " << p1.Nama << endl;
```

```
        cout <<"Umur: " << p1.Usia << endl;
        cout << "Gaji: " << p1.Gaji;

        return 0;
            }
```

Output dari Program diatas

```
Masukan Nama Lengkap: Abdul Ghani
Masukkan Usia: 28
Masukkan Gaji: 8000000

Menampilkan Informasi.
Nama: Abdul Ghani
Umur: 28
Gaji  : 8000000
```

Pada program diatas kita dapatkan informasi bahwa struktur pribadi yang memiliki tiga anggota yaitu, nama, usia dan gaji.

Di dalam fungsi main (), variabel struktur p1 didefinisikan. Kemudian, pengguna diminta untuk memasukkan informasi dan data yang dimasukkan oleh pengguna ditampilkan.

Variabel struktur dapat diteruskan ke fungsi dan dikembalikan dengan cara yang sama seperti argumen normal.

Meneruskan struktur ke fungsi di C ++

Variabel struktur dapat diteruskan ke fungsi dengan cara yang sama seperti argumen normal.

Contoh 1.

Listing Program Struktur dan Fungsi C++

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

struct Pribadi
{
   char nama[50];
   int usia;
   float gaji;
};
```

```
void displayData(Pribadi);   // Deklarasi Fungsi

int main()
{
   Pribadi p;

   cout << "Masukkan Nama Lengkap: ";
   cin.get(p.nama, 50);
   cout << "Masukkan Usia: ";
   cin >> p.usia;
   cout << "Enter Gaji: ";
   cin >> p.gaji;

   // Memanggil Fungsi dengan Struktur Variabel sebagai argumen
   displayData(p);

   return 0;
}

void displayData(Pribadi p)
{
   cout << "\nMenampilkan Informasi." << endl;
   cout << "Nama: " << p.nama << endl;
   cout <<"Usia: " << p.usia << endl;
   cout << "Gaji: " << p.gaji;
}
```

Output dari hasil program diatas

```
Masukkan Nama Lengkap: Bachtiar
Masukkan Usia: 33
Enter Gaji: 9000000

Menampilkan Informasi.
Nama: Bachtiar
Usia: 33
Gaji: 9000000
```

Contoh 3

Listing Program

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

struct Pribadi {
    char nama[50];
    int usia;
    float gaji;
};

Person getData(Pribadi);
void displayData(Pribadi);

int main()
{

    Pribadi p;

    p = getData(p);
    displayData(p);

    return 0;
}

Pribadi getData(Pribadi p) {

    cout << "Masukkan Nama Lengkap: ";
    cin.get(p.nama, 50);

    cout << "Masukkan Usia: ";
    cin >> p.usia;

    cout << "Masukkan Gaji: ";
    cin >> p.gaji;

    return p;
}

void displayData(Pribadi p)
{
    cout << "\nMenampilkan Informasi." << endl;
    cout << "Nama: " << p.nama << endl;
    cout <<"Usia: " << p.usia << endl;
```

```
    cout << "Gaji: " << p.gaji;
}
```

Output dari program ini sama dengan program di atas contoh 2 Dalam program ini, variabel struktur p dari tipe struktur Pribadi didefinisikan di bawah fungsi main ().

Variabel struktur p diteruskan ke fungsi getData () yang mengambil masukan dari pengguna yang kemudian dikembalikan ke fungsi utama.

p = getData (p);

Catatan: Nilai semua anggota struktur variabel dapat ditetapkan ke struktur lain menggunakan operator penugasan = jika kedua struktur variabel memiliki tipe yang sama. Anda tidak perlu menetapkan setiap anggota secara manual.

Kemudian variabel struktur p diteruskan ke fungsi displayData (), yang menampilkan informasi.

Variabel Dinamis

Variabel dinamis : dapat disiapkan dan dihapus saat program sedang dijalankan. Perhatikan program berikut :

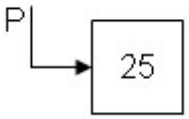
```
#include<iostream.h>
#include<stdlib.h>
void main(void)
{
        int *P;
        P=(int *)malloc(sizeof(int));
        *P=25;
        cout<<*P<<endl;
}
```

Keterangan : int *P : menyiapkan sebuah pointer P bertipe int

P = (int*)malloc(sizeof(int)) : menyiapkan suatu area bebas, tanpa nama, selebar sizeof(int), alamatnya dicatat dalam pointer P.

Untuk membebaskan area tersebut dapat digunakan instruksi : free(P)

Program diatas diilustrasikan sbb

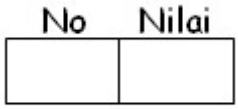
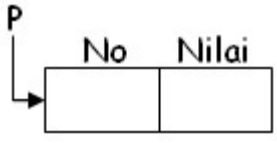
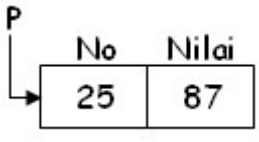


dapat :

Variabel struktur yang bersifat dinamis :

```
#include<iostream.h>
#include<stdlib.h>
void main(void)
{
 typedef struct{
        int No;
            int Nilai;
}strNode;
strNode *P;
P=(strNode*)malloc(sizeof(strNode));
       P->No=25;
            P->Nilai=87;
            cout<<"No = "<<P->No<<endl; cout<<"Nilai = "<<P-
>Nilai<<endl;
}
```

Program diatas dapat diilustrasikan sbb :

typedef struct{ int No; int Nilai; }strNode;	
P=(strNode*)malloc(sizeof(strNode))	
P->No=25; P->Nilai=87;	

C. SOAL LATIHAN/TUGAS

Latihan.3

1. Tulis program untuk menyimpan dan mencetak No urut, Nama, usia dan nilai mahasiswa menggunakan struktur.
2. Tulis program untuk menyimpan No urut. (mulai dari 1), nama dan umur 5 Mahasiswa kemudian cetak detail Mahasiswa dengan No urut 2.
3. Tulis program untuk menyimpan dan mencetak no. urut, nama, umur, alamat dan nilai 15 mahasiswa menggunakan struktur.
4. Kerjakan menu perpustakaan. Buat struktur yang berisi informasi buku seperti nomor akses, nama penulis, judul buku dan bendera untuk mengetahui apakah buku tersebut diterbitkan atau tidak.
5. Buat menu di mana hal berikut dapat dilakukan.

   1 - Menampilkan informasi buku

   2 - Tambahkan buku baru

   3 - Tampilkan semua buku di perpustakaan penulis tertentu

   4 - Menampilkan jumlah buku dengan judul tertentu

   5 - Menampilkan jumlah total buku di perpustakaan

   6 - Terbitkan buku

   (Jika kita menerbitkan buku, maka jumlahnya akan berkurang 1 dan jika kita menambahkan buku, jumlahnya bertambah 1)

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 4
SINGLE STACK

A. TUJUAN PEMBELAJARAN

Pada bab ini akan dijelaskan tentang mampu memahami konsep *Single Stack,* dan algoritma PUSH dan POP pada Single *Stack*

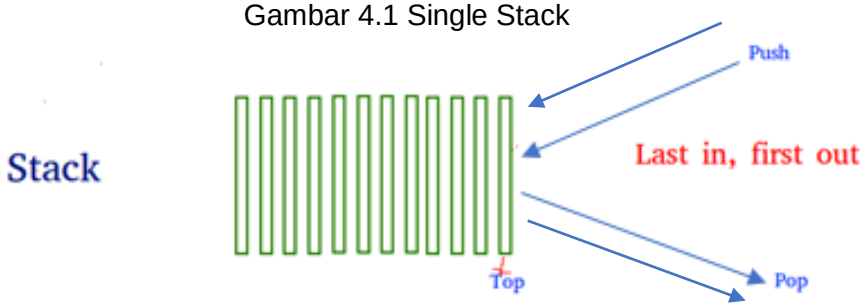
1. Mampu memahami konsep *Single Stack,* dan algoritma PUSH dan POP pada Single *Stack*

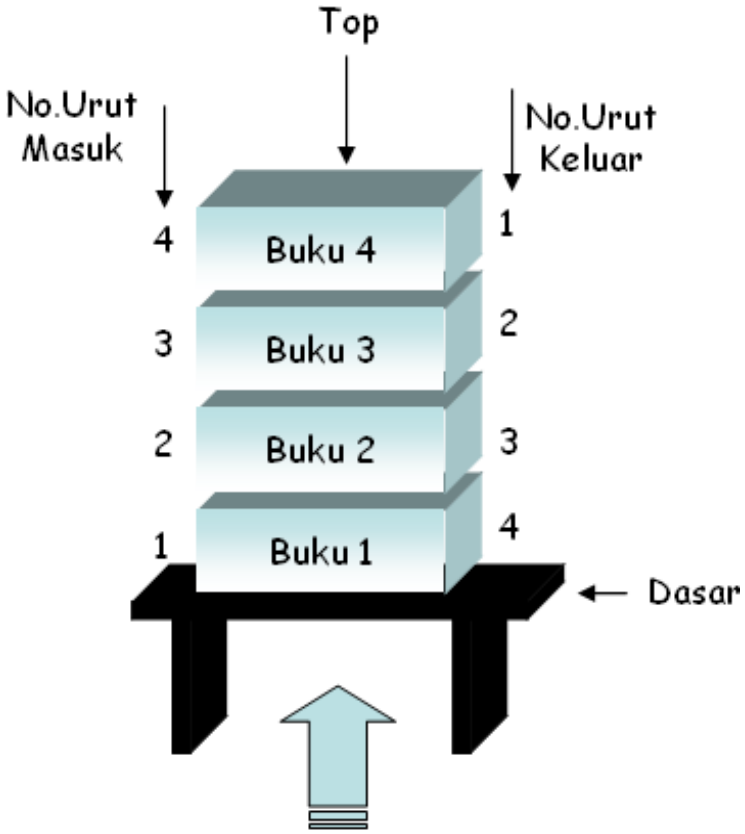
B. URAIAN MATERI

Pendahuluan

Stack adalah struktur data linier yang mengikuti urutan tertentu di mana operasi dilakukan. Urutannya mungkin *FILO (First In Last Out)* atau *LIFO (Last In First Out).*

Gambar 4.1 Single Stack





Bagaimana memahami tumpukan secara praktis?

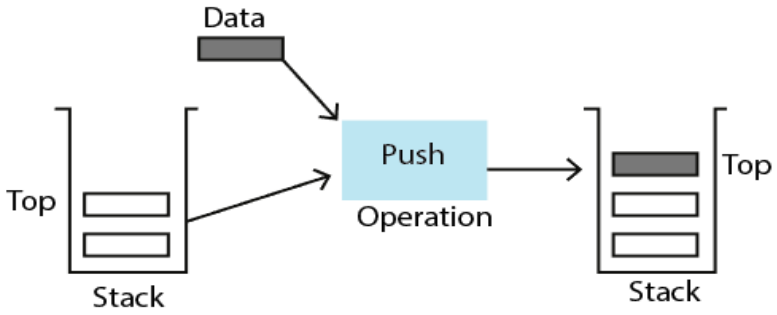
Ada banyak contoh tumpukan di kehidupan nyata. Pertimbangkan contoh sederhana piring yang ditumpuk satu sama lain di kantin. Pelat yang berada di atas adalah yang pertama harus dilepas, yaitu pelat yang telah ditempatkan di posisi paling bawah tetap berada di tumpukan untuk jangka waktu yang paling lama. Jadi, cukup terlihat mengikuti perintah LIFO / FILO.

Opearsi dasar *stack*

Ada dua operasi yang dasar yang bisa dilakukan dalam *stack,* yaitu :

1) PUSH

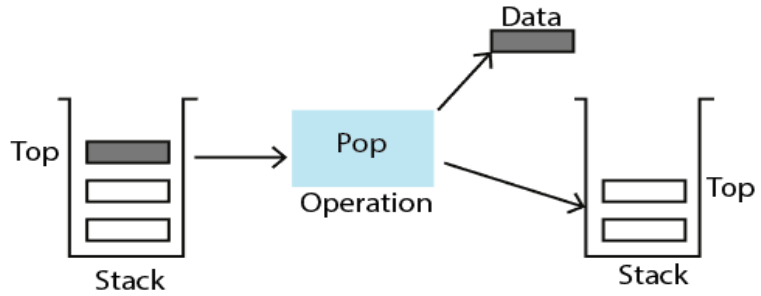
Operasi *push* terjadi apabila tumpukan dalam kondisi yang tidak dalam kondisi penuh lalu elemen pada urutan paling atas (terakhir) ditambahkan.



Gambar 4.2 Operasi Push

2) POP

Operasi *push* terjadi apabila tumpukan dalam kondisi yang tidak dalam kondisi kosong lalu elemen pada urutan paling atas (terakhir) diambil kemudian elemen tersebut dihapus dari stack.



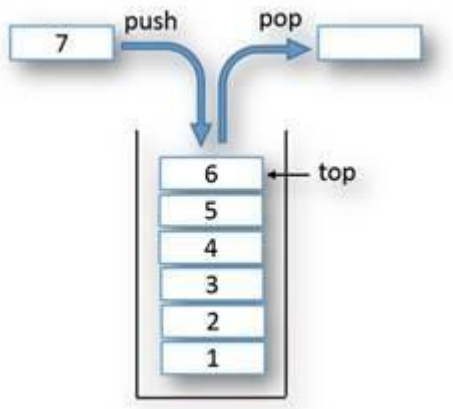
Gambar 4.3 Operasi Pop

Operasi lain dalam stack

Selain dari opearsi dasar di atas, ada beberapa operasi yang dapat di pakai dalam stack :

1. **Deklarasi** yaitu operasi pendeklarasian suatu stack.
2. **IsEmpty** yaitu operasi pemeriksa apa kondisi stack kosong.
3. **IsFull** yaitu operasi pemeriksa apa kondisi stack penuh.
4. **Inisialisasi** yaitu operasi pembuatan stack awal.

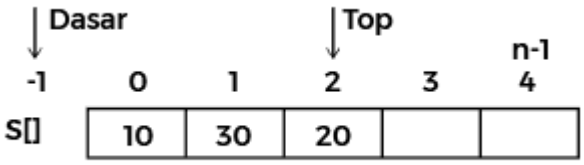
SINGLE STACK



Gambar 4.4 Proses dalam Single Stack

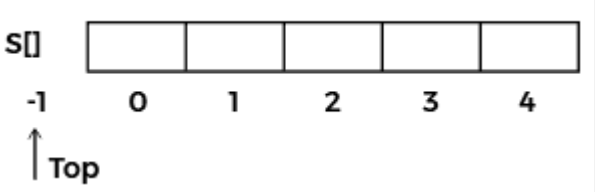
Dalam komputasi, stack adalah struktur data yang digunakan untuk menyimpan sekumpulan objek. Item individual dapat ditambahkan dan disimpan dalam stack menggunakan operasi push. Objek dapat diambil menggunakan operasi pop, yang menghapus item dari stack.

Ilustrasi single stack S[n] dengan n = 5.



Kondisi Stack

1. S[n]

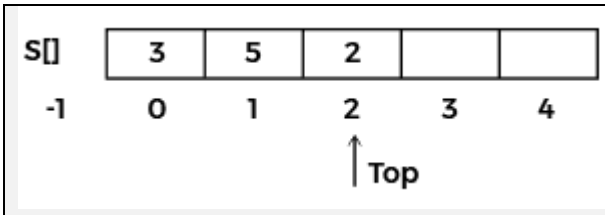


Gambar 4.5 Illustrasi Single Stack (a)

Kosong : Top = -1

Bisa Diisi : Top < n -1

2. S[n]

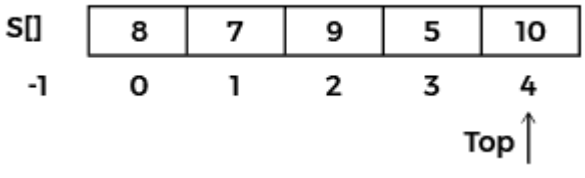


Gambar 4.6 Illustrasi Single Stack (b)

Ada isinya : Top > -1

Masih dapat diisi : Top < n -1

3. S[n]



Gambar 4.7 Illustrasi Single Stack (c)

Penuh : Top = n – 1

Ada isinya : Top > n-1

1. PROSES SINGLE STACK

Proses Single Stack ada 3 macam yaitu:

a. AWAL (inisialisasi)
b. PUSH (Masuk atau Insert)
c. POP (Keluar atau Delete)

1) Proses AWAL (Inisialisasi)

Top = -1;

```
Void AWAL ()
        { Top =-1;
        }
```

2) PUSH (Masuk atau Insert)

```
Void PUSH ()
      {
         Top  =  Top  +
1;
         S[Top] = x ;
      }
```

3) POP (Keluar atau Delete)

```
Void POP ()
      {
         X  = S[Top] ;
         Top = Top - 1;
      }
```

2. KONDISI SINGLE STACK

Tabel 2.1 Kondisi Single Stack

No	Kondisi Stack	Ciri
1.	KOSONG tidak ada isinya (empty)	Top1 = -1
2.	PENUH tak bisa diisi lagi (full)	Top2 = n - 1
3.	BISA DIISI (kebalikan dari PENUH) / (not full)	Top2 < n - 1
4.	ADA ISINYA (kebalikan dari KOSONG) / (not empty)	Top2 > -1

Listing program Menerapkan Stack menggunakan Array

```
/* Program C ++ untuk mengimplementasikan operasidasar pada Stack */
#include <bits/stdc++.h>
using namespace std;
#define MAX 1000
class Stack {
    int top;
public:
    int a[MAX]; // Ukuran maksimum Stack
    Stack() { top = -1; }
    bool push(int x);
    int pop();
    int peek();
    bool isEmpty();
};
bool Stack::push(int x)
{
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow";
        return false;
    }
    else {
        a[++top] = x;
        cout << x << " didorong ke dalam tumpukan\n";
        return true;
    }
}

int Stack::pop()
{
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    }
    else {
        int x = a[top--];
        return x;
    }
}
int Stack::peek()
{
    if (top < 0) {
        cout << "Tumpukan Kosong";
        return 0;
```

```
    }
    else {
        int x = a[top];
        return x;
    }
}

bool Stack::isEmpty()
{
    return (top < 0);
}
// Program untuk menguji fungsi di atas
int main()
{
    class Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    cout << s.pop() << " Dikeluarkan dari tumpukan\n";

    return 0;
}
```

Hasil dari program diatas yaitu

```
10 didorong ke dalam tumpukan
20 didorong ke dalam tumpukan
30 didorong ke dalam tumpukan
30 Dikeluarkan dari tumpukan
```

Kelebihan:

- Mudah diimplementasikan. Memori disimpan karena penunjuk tidak terlibat.

Kekurangan:

- Ini tidak dinamis. Itu tidak tumbuh dan menyusut tergantung pada kebutuhan pada waktu proses

C. SOAL LATIHAN/TUGAS

Latihan 1.

1. Susunlah sebuah program C++ untuk menyiapkan array 1 dimensi yang digunakan sebagai Stack S sebanyak 10 elemen, bertipe integer. Kemudian lakukan proses simpan data ke stack (PUSH) atau proses mengeluarkan isi stack (POP) dengan proses sebagai berikut:
   1) Inputkan data dari keyboard. Bila data yang diinputkan bernilai 999, maka proses selesai.
   2) Bila data yang diinput bernilai >= 60, maka periksa kondisi stack. Bila stack masih bisa diisi, maka simpan data tersebut (PUSH) kedalam stack, dan proses diulang kembali mulai no 1. Tapi bila stack sudah penuh, data tidak jadi disimpan, kemudian cetak perkataan “Stack Penuh”, dan proses selesai.
   3) Bila data yang diinputkan <60, maka periksa kondisi stack. Bila stack ada isinya, maka ambil isi stack dan cetak ke layar, kemudian proses diulang kembali mulai no. 1. Bila stack taka da isinya, maka cetak perkataan “ Stack Kosong”, dan proses selesai.

2. Mengisi (PUSH) stack sampai PENUH, dan mengambil (POP) isi stack sampai KOSONG> Susunlah program C++ untuk menyiapkan array 1 dimensi yang digunakan sebagai Stack S sebanyak 10 elemen bertipe integer. Kemudian inputkan data (nilai numerik) dan simpan (PUSH) ke stack S. Proses input dan PUSH selesai bila data yang diinputkan bernilai = 999, atau Stack S Penuh. (Nilai 999 dipakai sebagai end of data, tidak ikut diPush ke stack S). Setelah itu keluarkan (POP) isi stack dan cetak ke layar satu per satu sampai stack menjadi kosong.
3. Tulis program (potongan program) untuk menginput data melalui keyboard satu persatu dan mem Push data tersebut ke Stack sampai stack penuh tak bisa diisi lagi
4. Tulis program (potongan program) untuk mengeluarkan (POP) isi Stack satu persatu dan mencetaknya sampai stack menjadi kosong.
5. Tulislah ciri dari single stack untuk kondisi sebagai berikut:
   a. Kosong
   b. Penuh
   c. Bisa diisi
   d. Ada isinya

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 5
DOUBLE STACK

A. TUJUAN PEMBELAJARAN

Bab ini akan menjelaskan dan memahami konsep *Double Stack,* dan algoritma PUSH dan POP pada double *Stack*:

1. Mampu dan memahami konsep Double Stack, dan algoritma PUSH dan POP pada double Stack

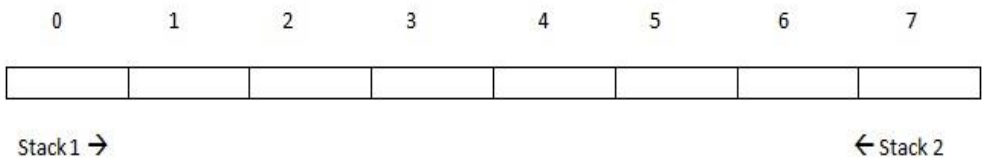
B. URAIAN MATERI

Buat struktur data double Stack yang mewakili dua tumpukan. Implementasi double Stack harus menggunakan hanya satu array, yaitu, kedua tumpukan harus menggunakan array yang sama untuk menyimpan elemen.

Double Stack

Apa itu Double Stack?

Kami juga dapat mempertahankan dua tumpukan dalam satu array. Jenis implementasi ini dikenal sebagai double stack. Dalam implementasi ini kedua tumpukan tumbuh berlawanan arah. Satu dari indeks yang lebih rendah ke indeks yang lebih tinggi dan yang kedua dari indeks yang lebih tinggi ke indeks yang lebih rendah.



TOP1 awalnya adalah -1 dan TOP2 awalnya adalah MAX.

Listing Program Deklarasi Double Stack

```
#define MAX 5

//Deklarasi Double Stack
class DStack
{
   private:
      int top1;
      int top2;
      int ele[MAX];

   public:
      DStack();
      void pushA(int  item);
      void pushB(int  item);
      int  popA (int *item);
      int  popB (int *item);

};
```

Proses :

a) AWAL(Inisialisasi)

b) PUSH1, Push untuk stack1

c) POP1, Pop untuk stack1

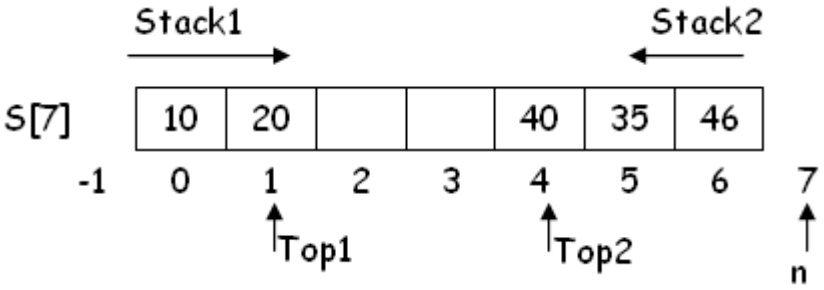
d) PUSH2, Push untuk stack2

e) POP2, Pop untuk stack

a. Fungsi dasar proses AWAL :

```
void AWAL(void)
{
```

1. ILUSTRASI



2. PROSES DOUBLE STACK

Proses pada double Stack ada 5 macam yaitu:

a. AWAL (inisialisasi)
b. PUSH1, Push untuk Stack1 (Masuk atau Insert)
c. POP1, Pop untuk Stack1 (Keluar atau Delete)
d. PUSH2, Push untuk Stack2 (Masuk atau Insert)
e. POP2, Pop untuk Stack2 (Keluar atau Delete)

1) Proses AWAL (Inisialisasi)

Top1 = -1;

Top2 = n

```
Void AWAL ()
        {
          Top1 =-1;
          Top2 =n;
        }
```

2) Algoritma dasar untuk PUSH1 (mengisi stack1)

```
Void PUSH1 ()
        {
           Top1 = Top1 + 1;
           S[Top1] = x ;
        }
```

3) Algoritma dasar untuk POP1 (mengambil isi stack1)

```
Void POP1 ()
        {
           X  = S[Top1] ;
           Top1 = Top1 - 1;
        }
```

4) Algoritma dasar untuk PUSH2 (mengisi stack2)

```
Void PUSH1 ()
        {
           Top2 = Top2 - 1;
           S[Top2] = x ;
        }
```

5) Algoritma dasar untuk POP2 (mengambil isi stack2)

```
Void POP1 ()
        {
           X  = S[Top2] ;
           Top2 = Top2 - 1;
        }
```

3. KONDISI DOUBLE STACK

Tabel 5.1 Kondisi Double Stack

No	Kondisi Stack	Ciri
1.	Stack1 KOSONG	Top1 = -1
2.	Stack2 KOSONG	Top2 = n
3.	Stack PENUH Baik Stack1 maupun Stack2 TIDAK BISA DIISI	Top2 – Top1 = 1
4.	Stack BISA DIISI	Top2 – Top1 > 1

	Baik Stack1 maupun Stack2 BISA DIISI	
5.	Stack1 ADA ISINYA	Top1 > -1
6.	Stack2 ADA ISINYA	Top2 < n

Program lengkap untuk mengimplementasikan Double Stack

```
#include <iostream>
using namespace std;
#define MAX 5

//Deklarasi Double Stack
class DStack
{
   private:
      int top1;
      int top2;
      int ele[MAX];

   public:
      DStack();
      void pushA(int  item);
      void pushB(int  item);
      int  popA (int *item);
      int  popB (int *item);

};

//IInisialisasi Double Stack
DStack::DStack()
{
   top1 = -1;
   top2 = MAX;
}

//Operasi Push di Stack1
void DStack::pushA( int item )
{
   if( top2 == top1 + 1 )
   {
      cout<<"\nStack Overflow Stack1";
      return;
   }
```

```
    top1++;
    ele[top1] = item;

    cout<<"\nItem disisipkan di Stack1 : "<< item;
}

//Operasi Push di Stack2
void DStack::pushB( int item )
{
    if( top2 == top1 + 1 )
    {
        cout<<"\nStack Overflow Stack2";
        return;
    }

    top2--;
    ele[top2] = item;

    cout<<"\nItem disisipkan di Stack2 : "<< item;
}

//Operasi Pop di Stack1
int DStack::popA( int *item )
{
    if( top1 == -1 )
    {
        cout<<"\nStack Underflow Stack1";
        return -1;
    }

    *item = ele[top1--];
    return 0;
}

//Operasi Pop di Stack2
int DStack::popB( int *item )
{
    if( top2 == MAX )
    {
        cout<<"\nStack Underflow Stack2";
        return -1;
    }

    *item = ele[top2++];
```

```
    return 0;
}


int main()
{
    int item = 0;

    DStack s = DStack();

    s.pushA(10);
    s.pushA(20);
    s.pushA(30);

    s.pushB(40);
    s.pushB(50);
    s.pushB(60);


    if( s.popA(&item) == 0 )
        cout<<"\nMenghapus Item dari Stack1 : "<< item;
    if( s.popA(&item) == 0 )
        cout<<"\nMenghapus Item dari Stack1 : "<< item;
    if( s.popA(&item) == 0 )
        cout<<"\nMenghapus Item dari Stack1 : "<< item;

    if( s.popB(&item) == 0 )
        cout<<"\nMenghapus Item dari Stack2 : "<< item;
    if( s.popB(&item) == 0 )
        cout<<"\nMenghapus Item dari Stack2 : "<< item;
    if( s.popB(&item) == 0 )
        cout<<"\nMenghapus Item dari Stack2 : "<< item;

    cout<< endl;

    return 0;
}
```

Hasil dari output program diatas

```
Item disisipkan di Stack1 : 10
Item disisipkan di Stack1 : 20
Item disisipkan di Stack1 : 30
Item disisipkan di Stack2 : 40
```

```
Item disisipkan di Stack2 : 50
Stack Overflow Stack2
Menghapus Item dari Stack1 : 30
Menghapus Item dari Stack1 : 20
Menghapus Item dari Stack1 : 10
Menghapus Item dari Stack2 : 50
Menghapus Item dari Stack2 : 40
Stack Underflow Stack2
```

C. SOAL LATIHAN/TUGAS

Latihan 5.

1. Tulislah algoritma yang lengkap untuk :
    a. Mengisi Stack1 (PUSH1)
    b. Menghapus isi Stack1 (POP1)
    c. Mengisi Stack2 (PUSH2)
    d. Menghapus isi Stack2 (POP2)

2. Tulislah Algoritma Dasar dari:
    a. Mengisi Stack1 (PUSH1)
    b. Menghapus isi Stack1 (POP1)
    c. Mengisi Stack2 (PUSH2)
    d. Menghapus isi Stack2 (POP2)
3. Sebutkan ciri double stack dari kondisi:
    a. Penuh. Baik Stack1 maupun Stack2 tak bisa diisi lagi
    b. Baik Stack1 maupun Stack2 bisa diisi lagi
    c. Baik Stack1 maupun Stack2 tak ada isinya
4. Tulis algoritma yang lengkap untuk mengambil isi Stack1 satu persatu dan mencetaknya ke layar, sampai stack1 isinya kosong
5. Tulis algoritma yang lengkap untuk mengambil isi Stack2 satu persatu dan mencetaknya ke layar monitor, sampai stack2 isinya kosong.

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 6
LINEAR QUEUE

A. TUJUAN PEMBELAJARAN

Pada pertemuan kali ini mahasiswa akan mampu memahami, mengilustrasikan dan mengimplementasikan algoritma Linier Queue.

1. Mampu memahami, mengilustrasikan dan mengimplementasikan algoritma Linier Queue

B. URAIAN MATERI

Seperti Stack , Queue adalah struktur linier yang mengikuti urutan tertentu di mana operasi dilakukan. Urutannya adalah First **I**n First **O**ut (FIFO). Contoh antrian yang baik adalah antrian konsumen mana pun untuk sumber daya di mana konsumen yang datang lebih dulu dilayani terlebih dahulu.

Perbedaan antara tumpukan dan antrian adalah dalam menghapus. Dalam tumpukan kami menghapus item yang paling baru ditambahkan; dalam antrian, kami menghapus item yang paling terakhir ditambahkan.

1. **Aplikasi Antrian:**

Queue digunakan ketika sesuatu tidak harus diproses dengan segera, tetapi harus diproses dengan urutan pertama seperti *Breadth First Search* . Properti Queue ini membuatnya juga berguna dalam skenario jenis berikut.

a. Ketika sumber daya dibagikan di antara banyak konsumen. Contohnya termasuk penjadwalan CPU, Penjadwalan Disk.

b. Ketika data ditransfer secara asinkron (data tidak harus diterima pada kecepatan yang sama seperti yang dikirim) antara dua proses. Contohnya termasuk Buffer IO, pipa, file IO, dll.

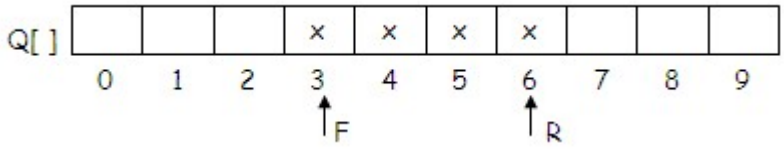
2. **Implementasi Array di Queue**

Untuk mengimplementasikan antrian, kita perlu melacak dua indeks, depan dan belakang. Kami mengantrekan item di belakang dan menghapus item dari depan. Jika kita hanya menaikkan indeks depan dan belakang, maka mungkin ada masalah, bagian depan mungkin mencapai akhir larik. Solusi untuk masalah ini adalah menaikkan bagian depan dan belakang secara melingkar .

3. **Linear Queue**

1) Ilustrasi

Misal n= 10



F (Front) : menunjuk pengantri paling depan/siap untuk keluar/ siap untuk dilayani

R (Rear) : menunjuk pengantri paling belakang/paling akhir masuk R = 6, artinya :

- Pernah masuk 7 pengantri dengan urutan masuk Q[0], Q[1], Q[2], Q[3], Q[4], Q[5], Q[6] F = 3, artinya :
- Sudah keluar sebanyak 3 pengantri dengan urutan keluar Q[0], Q[1], Q[2]

2) Prinsip : FIFO(First In First Out) atau FIFS (First In First Serve) III. Proses :

- AWAL (Inisialisasi)
- INSERT (Sisip, Masuk, Simpan, Tulis)
- DELETE (Hapus, Keluar, Ambil, Dilayani)
- RESET (Kembali ke keadaan awal)

a) Fungsi dasar untuk proses AWAL :

```
void AWAL(void)
{
F = 0;
R = -1;
}
```

b) Fungsi dasar proses INSERT :

```
void INSERT(void)
{
        R = R + 1;
        Q[R] = x;
}
```

atau

```
void INSERT(void)
{
        Q[++R] = x;
}
```

c) Fungsi dasar proses DELETE :

```
void DELETE(void)
{ x = Q[F]; F = F +
        1;
}
```

atau

```
void DELETE(void)
{ x = Q[F++];
}
```

d) Fungsi dasar proses RESET :

```
void RESET(void)
{
F = 0;
R = -1;
}
```

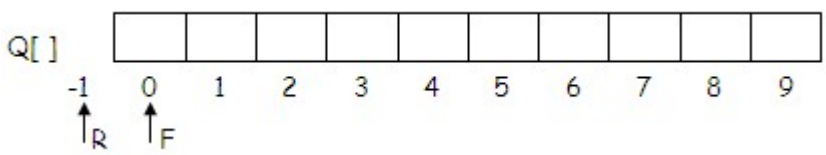
3) Kondisi antrian (n : jml elemen array)

a) Kondisi awal

a. F = 0, R = -1 kondisi awal

b. F = R + 1 antrian kosong

c. R < n - 1 antrian bisa diisi

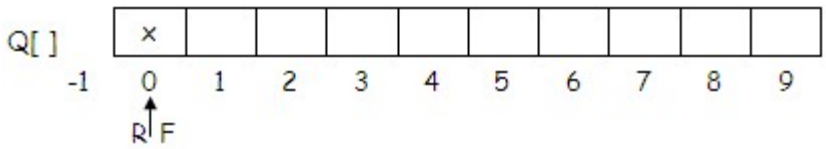


b) Misal masuk 1 pengantri, belum ada yang keluar

a. F = 0 belum ada yang keluar

b. F < R + 1 antrian ada isinya

c. R < n – 1 antrian bisa diisi



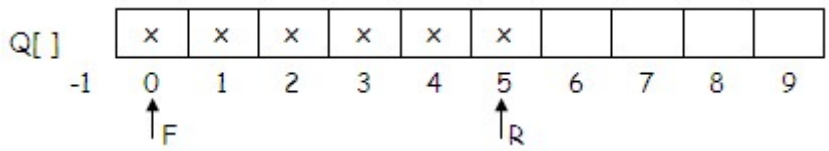
2) Misal masuk lagi 5 pengantri, belum ada yang keluar

a. R = 5 sudah pernah masuk 6

b. F = 0 belum ada yang keluar

c. F < R + 1 antrian ada isinya

d. R < n – 1 antrian bisa diisi



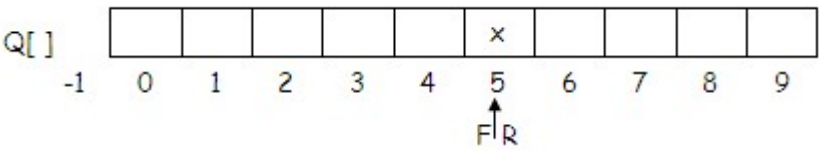
3) Misal keluar 5 pengantri

a. F = 5 sudah keluar 5 pengantri

b. F = R tinggal 1 pengantri

c. F < R + 1 antrian ada isinya

d. R < n – 1 antrian bisa diisi



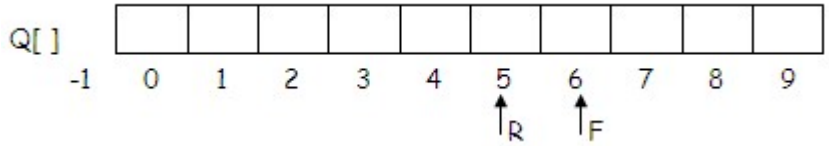
4) Misal keluar lagi satu pengantri

   e. F = 6 sudah keluar 6 pengantri

   f. R = 5

   g. F = R + 1 antrian kosong

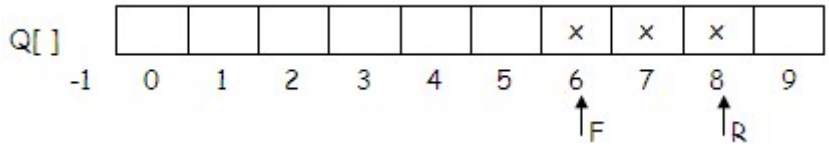
   h. R < n + 1 antrian bisa diisi



5) Misal masuk lagi 3 pengantri

   i. F < R + 1 antrian ada isinya

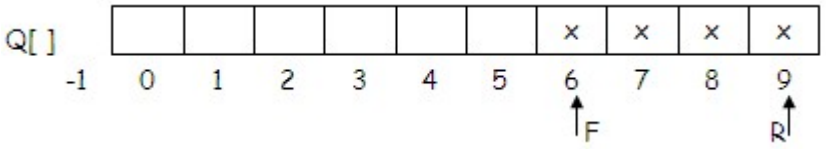
   j. R < n – 1 antrian masih bisa diisi



6) Misal masuk lagi 1 pengantri

   k. F < R + 1 antrian ada isinya

   l. R = n – 1 antrian penuh

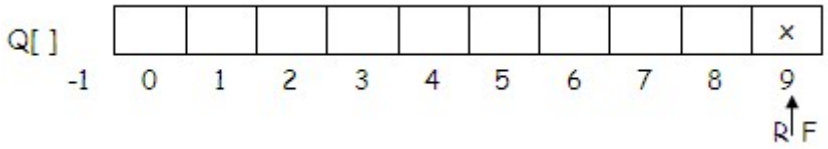


7) Misal keluar 3 pengantri

   a. F < R + 1 antrian ada isinya

   b. F = R antrian sisa 1 pengantri

c. R = n – 1 antrian penuh



8) Misalkan keluar 1 pengantri

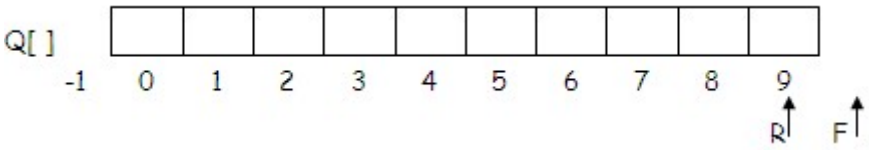
a. F = n semua antrian sudah keluar

b. F = R + 1 antrian kosong

c. R = n – 1 antrian penuh (disebut penuh)

d. F = R + 1 dan R = n – 1 kondisi khusus :

- Kosong karena tidak ada isinya
- Penuh karena tidak bisa diisi
- Perlu direset(kembali ke posisi awal)

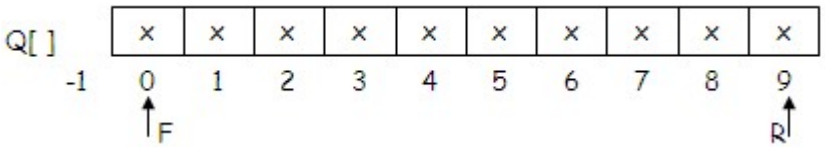


9) Kondisi khusus lainnya :

a. Antrian penuh tapi belum ada yang keluar

i. F = 0 belum ada yang keluar ii.

R = n – 1 antrian penuh



1. Kondisi antrian:

	Kondisi	Ciri
a	Kosong	F = R + 1 dimana saja

b	Penuh	R = n – 1
c	Bisa diisi	R < n – 1
d	Ada isinya	F < R + 1
e	Perlu direset	F = R + 1 dan R = n - 1

2. Fungsi INSERT dan DELETE lengkap

```
void INSERT(void)
{
    if(R < n – 1)
        Q[++R] = x;
    else
        cout<<”Antrian penuh”;
}
```

```
void DELETE(void)
{
    if(F < R + 1)
    { x = Q[F++]; if(F == n)
        {
            F = 0;
            R = -1;
        }
    }
    else cout<<”Antrian kosong”;
}
```

Kelebihan Implementasi Array:

- Mudah diimplementasikan.

Kontra Implementasi Array:

- Struktur Data Statis, ukuran tetap.
- Jika antrian memiliki sejumlah besar operasi enqueue dan dequeue, pada titik tertentu kita mungkin tidak dapat memasukkan elemen dalam antrian meskipun antriannya kosong (masalah ini dihindari dengan menggunakan antrian melingkar).

Listing program deklarasi linier queue

```
#define MAX 5
class Queue
{
   private:
      int front,rear;
      int ele[MAX];
   public:
      //Inisialisasi queue
      Queue()
      {
         front =  0;
         rear  = -1;
      }
      int  isFull();
      int  isEmpty();
      void insertQueue(int  item);
      int  deleteQueue(int *item);
};
```

Listing Program Implementasi Antrian Linear untuk semua operasi antrian di atas

```
#include <iostream>
using namespace std;
#define MAX 5
class Queue
{
   private:
      int front,rear;
      int ele[MAX];
   public:
      Queue() //Inisialisasi queue
      {
         front =  0;
         rear  = -1;
```

```
        }

        int  isFull();
        int  isEmpty();
        void insertQueue(int  item);
        int  deleteQueue(int *item);
};

// Untuk mengecek antrian sudah penuh atau belum
int Queue::isFull()
{
    int full = 0 ;

    if( rear == MAX-1 )
        full = 1;

    return full;
}

// Untuk memeriksa antrian kosong atau tidak
int Queue::isEmpty()
{
    int empty = 0 ;

    if( front == rear + 1 )
        empty = 1;

    return empty;
}


// Masukkan Item ke queue
void Queue:: insertQueue(int item)
{
    if( isFull() )
    {
        cout<<"\nQueue OverFlow" << endl;
        return;
    }

    ele[++rear]=item;
    cout<<"\nNilai yang disisipkan :" << item;
}

//hapus item dari queue
```

```
int Queue:: deleteQueue(int *item)
{
   if( isEmpty() )
   {
      cout<<"\nQueue Underflow" << endl;
      return -1;
   }

   *item = ele[front++];
   return 0;
}

int main()
{
   int item=0;

   Queue q = Queue();

   q.insertQueue(10);
   q.insertQueue(20);
   q.insertQueue(30);
   q.insertQueue(40);
   q.insertQueue(50);
   q.insertQueue(60);

   if(q.deleteQueue( &item)==0)
      cout<<"\nItem dihapus : "<< item;

   if(q.deleteQueue( &item)==0)
      cout<<"\nItem dihapus : "<< item;

   if(q.deleteQueue( &item)==0)
      cout<<"\nItem dihapus : "<< item;

   if(q.deleteQueue( &item)==0)
      cout<<"\nItem dihapus : "<< item;

   if(q.deleteQueue( &item)==0)
      cout<<"\nItem dihapus : "<< item;

   if(q.deleteQueue( &item)==0)
      cout<<"\nItem dihapus : "<< item;

   cout<< endl;
   return 0;
```

```
}
```

Hasil dari pemograman diatas

```
Nilai yang disisipkan :10
Nilai yang disisipkan :20
Nilai yang disisipkan :30
Nilai yang disisipkan :40
Nilai yang disisipkan :50
Queue OverFlow

Item dihapus : 10
Item dihapus : 20
Item dihapus : 30
Item dihapus : 40
Item dihapus : 50
Queue Underflow
```

C. SOAL LATIHAN/TUGAS

Latihan 6

1. Tulis algoritma yang lengkap untuk mengisi antrian record per record sebanyak 10 record selama antrian belum penuh. Apabila antrian penuh, walaupun belum mengisi 10 record, proses pengisian dihentikan.
2. Tulis algoritma yang lengkap untuk mendelete antrian record per record sebanyak 10 record selama antrian masih ada isinya. Bila antrian sudah kosong, walaupun belum mendelete sebanyak 10 record, proses delete dihentikan.
3. Tulis algoritma dasar untuk:
    a. Inisialisasi
    b. Insert sebuah record
    c. Delete sebuah record
    d. Reset
4. Sebutkan ciri dari Linier Queue untuk kondisi:
    a. Kosong taka da isinya
    b. Penuh, tak bisa diisi
    c. Bisa diisi

d. Ada isinya
e. Antrian tak bisa diisi lagi, tapi belum ada isi antrian yang sudah keluar atau sudah dilayani
f. Antrian perlu direset

5. Jika n = 100, maka untuk:
   a. Bila F = 15, dan R = 37, maka jumlah pengantri yang belum dilayani = ……..
   b. Bila F = 15, dan R = 37, maka jumlah kolom yang masih bisa diisi = ……..

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 7
CIRCULAR QUEUE

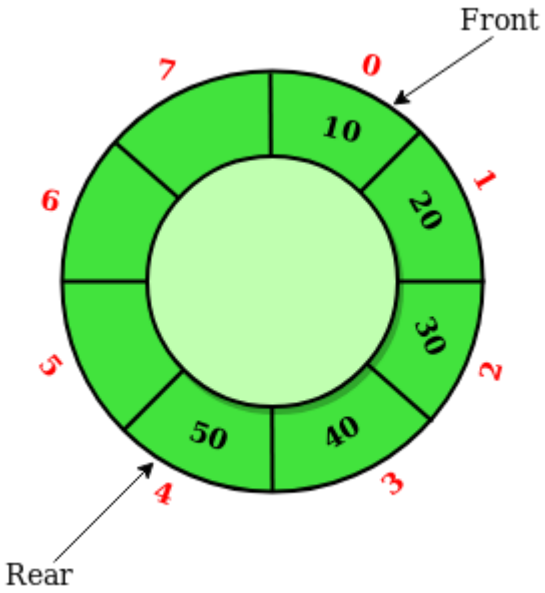
A. TUJUAN PEMBELAJARAN

Pada bab ini akan dijelaskan konsep memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Circular Queue*

1. Mampu memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Circular Queue*

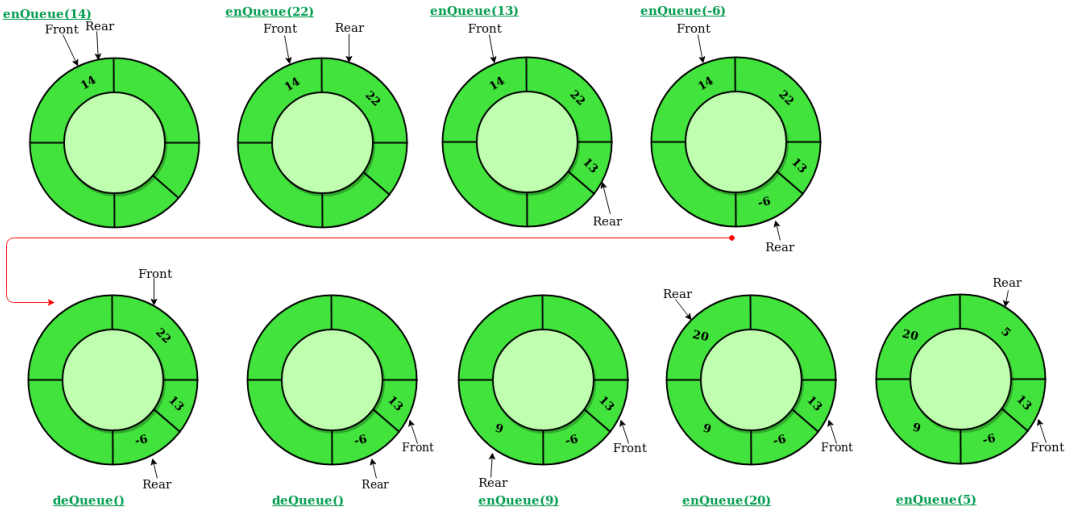
B. URAIAN MATERI

Circular Queue adalah struktur data linier di mana operasi dilakukan berdasarkan prinsip FIFO (First In First Out) dan posisi terakhir terhubung kembali ke posisi pertama untuk membuat lingkaran. Hal ini juga disebut 'Ring Buffer'



Gambar 7.1 Circular Queue

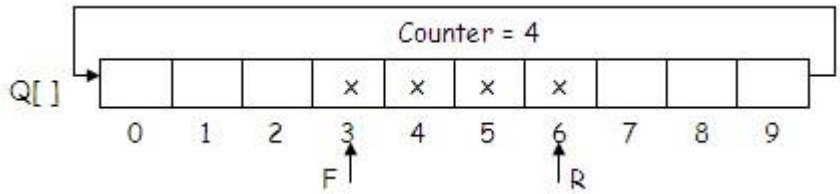
Dalam Queue normal, kita dapat memasukkan elemen sampai antrian menjadi penuh. Tapi begitu antrian menjadi penuh, kita tidak dapat memasukkan elemen berikutnya bahkan jika ada ruang di depan antrian.



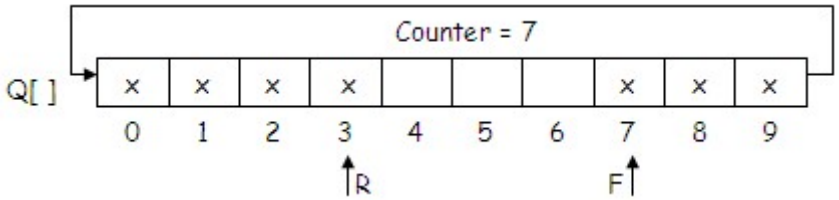
Gambar 7.2 Proses Queue

1. **Representasi Cicular Queue**

Misal n= 10



atau



Counter : Jumlah pengantri yang ada dalam antrian

2. **Definisi :**

```
#define n 10 int Q[n];
```

F tidak selalu <= R

Setelah R dan F sampai ke n-1, maka tidak direset tetapi melingkar ke

3. **Prinsip : FIFO(First In First Out) atau III. Proses :**

- AWAL (Inisialisasi)
- INSERT (Sisip, Masuk, Simpan, Tulis)
- DELETE (Hapus, Keluar, Ambil, Dilayani)

a) Fungsi dasar untuk proses AWAL :

```
void AWAL(void)
{
        F = 0;
        R = -1;
    COUNTER = 0
}
```

b) Fungsi dasar proses INSERT :

```
void INSERT(void)
{
        R = (R + 1) % n;
        Q[R] = x;
      COUNTER++;
}
```

c) Fungsi dasar proses DELETE :

```
void DELETE(void)
{
x = Q[F]; F =(F + 1) % n;
        COUNTER--;
}
```

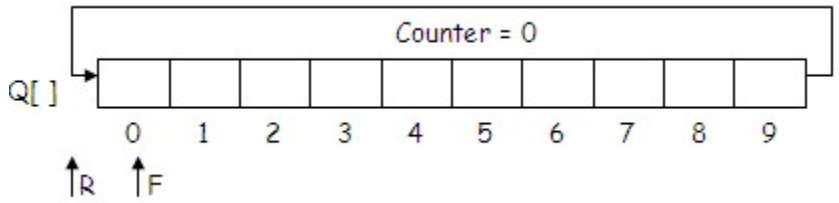
4. **Kondisi antrian (n : jumlah elemen array).**

Beberapa kondisi antrian al :

1) Kondisi awal

a. F = 0, R = -1, COUNTER = 0 kondisi awal

b. COUNTER = 0 Antrian kosong

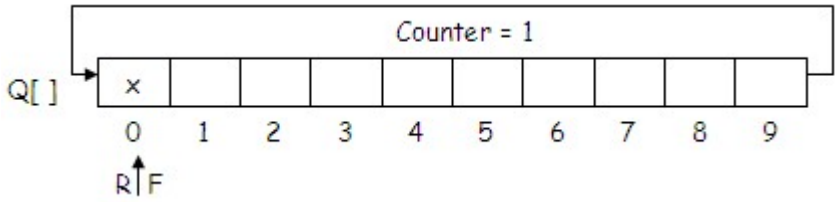


2) Misal masuk 1 pengantri, belum ada yang keluar

a. COUNTER > 0 ada isinya

b. F = R isinya hanya 1

c. Counter < n masih bisa diisi

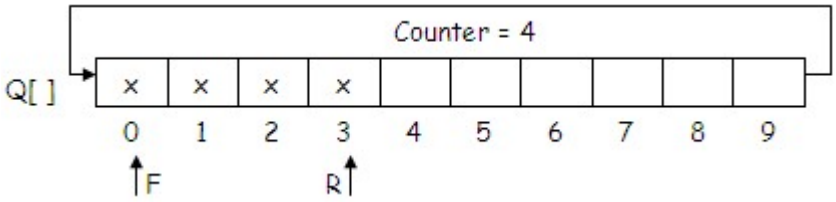


3) Misal masuk lagi 3 pengantri, belum ada yang keluar

a. COUNTER > 0 ada isinya

b. F <> R + 1 ada isinya

c. COUNTER < n masih bisa diisi

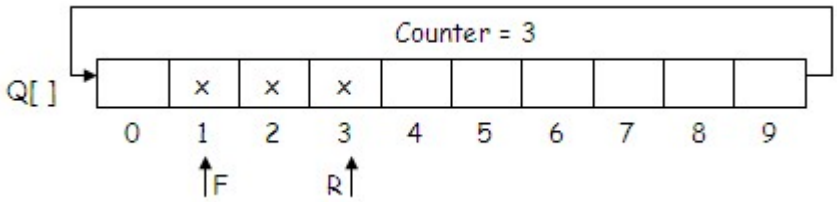


4) Misal keluar 1 pengantri

a. COUNTER > 0 ada isinya

b. F <> R + 1 ada isinya

c. COUNTER < n bisa diisi

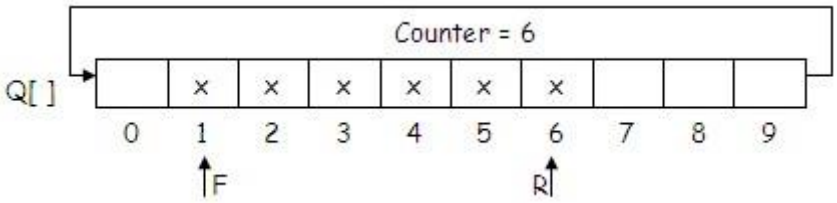


5) Misal masuk lagi 3 pengantri

a. COUNTER > 0 ada isinya

b. F <> R + 1 ada isinya

c. COUNTER < n bisa diisi

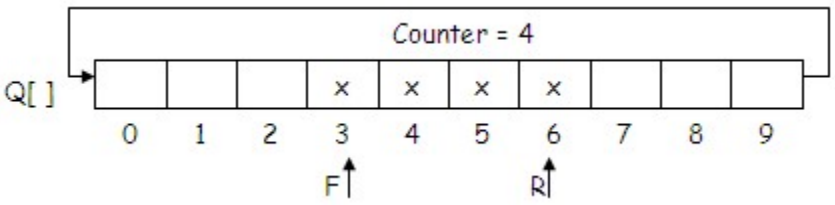


6) Misal keluar 2 pengantri

a. COUNTER > 0 ada isinya

b. R <> R + 1 ada isinya

c. COUNTER < n bisa diisi

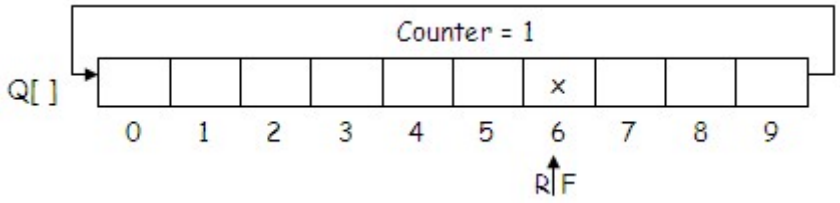


7) Misal keluar lagi 3 pengantri

a. COUNTER > 0 ada isinya

b. F = R hanya 1 pengantri

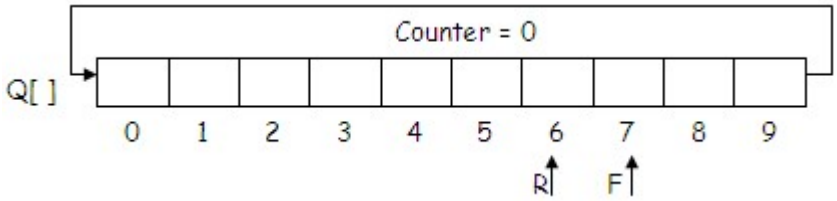
c. COUNTER < n bisa diisi



8) Misal keluar lagi 1 pengantri

a. COUNTER = 0 antrian kosong

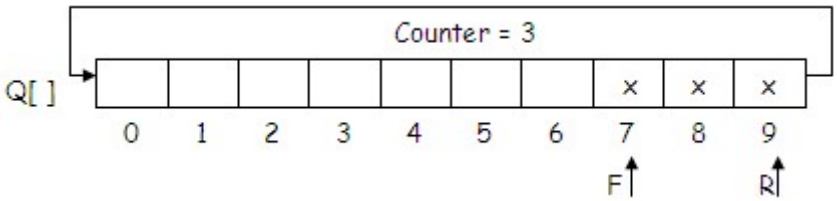
b. COUNTER < n bisa diisi



9) Misalkan masuk lagi 3 pengantri

a. COUNTER > 0 ada isinya

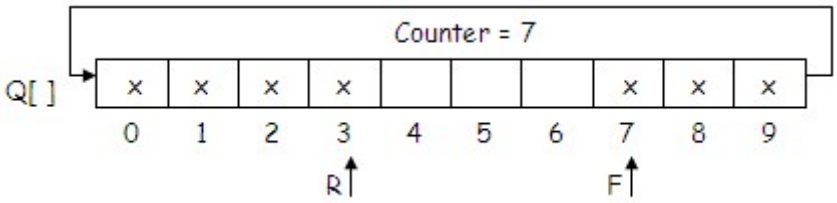
b. COUNTER < n bisa diisi



10) Misal masuk lagi 4 pengantri

a. COUNTER > 0 ada isinya

b. COUNTER < n bisa diisi

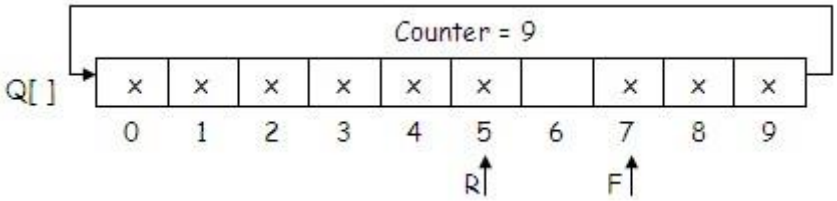


11) Misal masuk lagi 2 pengantri

a. COUNTER >0 ada isinya

b. F <> R + 1 ada isinya

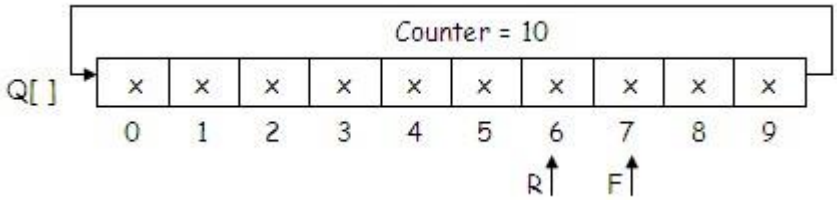
c. COUNTER < n bisa diisi



12) Misal masuk lagi 1 pengantri

a. COUNTER > 0 ada isinya

b. COUNTER = n antrian penuh



5. **Kondisi antrian:**

No	**Kondisi antrian**	**Ciri**
a	Kosong	COUNTER = 0
b	Penuh	COUNTER = n
c	Bisa diisi	COUNTER < n
d	Ada isinya	COUNTER > 0

6. **Fungsi INSERT dan DELETE lengkap**

```
void INSERT(void)
{
        if(COUNTER < n)
        {
                R = (R + 1) % n;
                 Q[R] = x;
                COUNTER++;
        }
        else cout<<"Antrian penuh";
}
```

```
void DELETE(void)
{
        if(COUNTER > 0)
        { x = Q[F]; F = (F + 1)
                %n;
                 COUNTER--;
        }
        else cout<<"Antrian kosong";
}
```

Listing Program Circular Queue

```
#define MAX 5

//Deklarasi Circular Queue
class CirQueue
{
   private:
      int front   ;
      int rear    ;
      int count   ;
      int ele[MAX]    ;

   public:
      CirQueue();
      int  isFull();
      int  isEmpty();
      void insertQueue(int item);
      int  deleteQueue(int *item);
};
```

Listing Program untuk implementasi Circular Queue untuk semua operasi antrian di atas

```
#include <iostream>

using namespace std;
#define MAX 5

//Deklarasi Circular Queue
class CirQueue
{
   private:
      int front   ;
      int rear    ;
      int count   ;
      int ele[MAX]    ;

   public:
      CirQueue();
      int  isFull();
      int  isEmpty();
      void insertQueue(int item);
      int  deleteQueue(int *item);
};
```

```
//Inisialisasi Circular Queue
CirQueue:: CirQueue()
{
   front =  0;
   rear  = -1;
   count =  0;
}

//Untuk mengecek queue sudah penuh atau belum
int CirQueue:: isFull()
{
   int full=0;

   if( count == MAX )
      full = 1;

   return full;
}

//Untuk memeriksa antrian kosong atau tidak
int CirQueue:: isEmpty()
{
   int empty=0;

   if( count == 0 )
      empty = 1;

   return empty;
}

//Sisipkan item ke dalam antrian melingkar
void CirQueue:: insertQueue(int item)
{
   if( isFull() )
   {
      cout<<"\nQueue Overflow";
      return;
   }

   rear= (rear+1) % MAX;

   ele[rear] = item;
   count++;

   cout<<"\ndimasukkan item : "<< item;
}

//Hapus item dari circular queue
```

```
int CirQueue:: deleteQueue(int *item)
{
   if( isEmpty() )
   {
      cout<<"\nQueue Underflow";
      return -1;
   }
   *item = ele[front];
   front = (front+1) % MAX;
   count--;
   return 0;
}

int main()
{
   int  item;
   CirQueue q = CirQueue();

   q.insertQueue(10);
   q.insertQueue(20);
   q.insertQueue(30);
   q.insertQueue(40);
   q.insertQueue(50);
   q.insertQueue(60);

   if( q.deleteQueue(&item) == 0 )
   {
      cout<<"\nItem dihapus:"<< item<< endl;
   }

   if( q.deleteQueue(&item) == 0 )
   {
      cout<<"\nItem dihapus:"<< item<< endl;
   }

   if( q.deleteQueue(&item) == 0 )
   {
      cout<<"\nItem dihapus:"<< item << endl;
   }

   if( q.deleteQueue(&item) == 0 )
   {
      cout<<"\nItem dihapus:"<< item<< endl;
   }

   if( q.deleteQueue(&item) == 0 )
   {
      cout<<"\nItem dihapus:"<< item<< endl;
```

```
    }

    q.insertQueue(60);
    if( q.deleteQueue(&item) == 0 )
    {
        cout<<"\nItem dihapus:"<< item<< endl;
    }

    if( q.deleteQueue(&item) == 0 )
    {
        cout<<"\nItem dihapus:"<< item<< endl;
    }

    cout<<"\n";
    return 0;
}
```

Hasil output dari program diatas

```
dimasukkan item : 10
dimasukkan item : 20
dimasukkan item : 30
dimasukkan item : 40
dimasukkan item : 50
Queue Overflow
Item dihapus:10

Item dihapus:20

Item dihapus:30

Item dihapus:40

Item dihapus:50

dimasukkan item : 60
Item dihapus:60

Queue Underflow
```

C. SOAL LATIHAN/TUGAS

Latihan 7

1. Sebutkan ciri circular Queue dalam kondisi:
    a. Kosong
    b. Penuh
    c. Bisa diisi
    d. Ada isinya
    e. Hanya berisi 10 record
    f. Tempat yang kosong hanya ada 10 tempat.
2. Tulis algoritma lengkap untuk:
    a. Insert sebuah record
    b. Delete sebuah record
3. Tulis algoritma yang lengkap untuk mengisi antrian record per record sebanyak 10 record selama antrian belum penuh. Apabila antrian penuh, walaupun belum mengisi 10 record, proses pengisian dihentikan.
4. Tulis algoritma yang lengkap untuk mendelete isi antrian record per record sebanyak 10 record selama antrian masih ada isinya. Apabila antrian sudah kosong, walaupun belum mendelete sebanyak 10 record, maka proses delete dihentikan.
5. Tulis untuk menghitung dan mencetak jumlah tempat(elemen) yang ada isinya bila diketahui nilai F dan R tanpa mengetahui nilai Counter.

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 8
DOUBLE ENDED QUEUE

A. TUJUAN PEMBELAJARAN

Pada bab ini akan dijelaskan tentang konsep memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Double Ended Queue*:

1. Mampu memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Double Ended Queue*

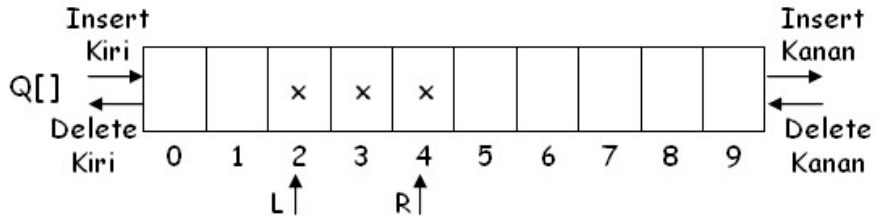
B. URAIAN MATERI

Deque atau Double Ended Queue adalah versi umum struktur data antrian yang memungkinkan memasukkan dan menghapus di kedua ujungnya.

Double Ended Queue

1. Representasi

Misal n= 10



Insert Kiri : masuk dari pintu kiri

Insert Kanan : masuk dari pintu kanan

Delete Kiri : keluar dari pintu kiri

Delete Kanan : keluar dari pintu kanan

2. Prinsip : Keluar masuk dari kedua ujung III. Proses :
   - AWAL (Inisialisasi)
   - INSERT (Sisip, Masuk, Simpan, Tulis)
   - DELETE (Hapus, Keluar, Ambil, Dilayani)

a) Fungsi dasar untuk proses AWAL :

```
void AWAL(void)
{
        L = 0;
        R = -1;
}
```

b) Fungsi dasar proses INSERT KIRI:

```
void INSERT_KIRI(void)
{
        Q[--L] = x;
}
```

c) Fungsi dasar proses INSERT KANAN:

```
void INSERT_KANAN(void)
{
        Q[++R] = x;
}
```

d) Fungsi dasar proses DELETE KIRI:

```
void DELETE_KIRI(void)
{
       x =
        Q[L++];
}
```

a) Fungsi dasar proses DELETE KANAN :

```
void DELETE_KANAN(void)
{
        x = Q[R--];
}
```

3. Kondisi antrian (n : jml elemen array).

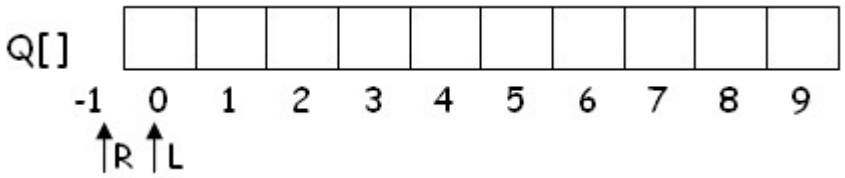
1) Kondisi awal

a. L = 0, R = -1 kondisi awal

b. L = R - 1 Antrian kosong

c. L = 0 Penuh kiri

d. R < n – 1 Bisa insert kanan

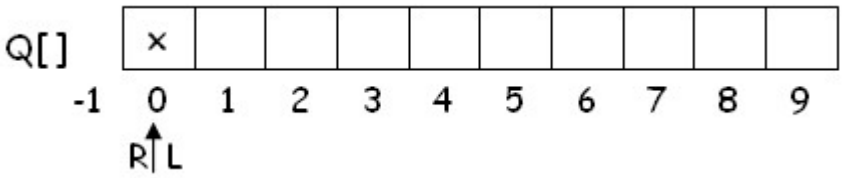


2) Misal masuk 1 pengantri dari kanan

a. L < R + 1 ada isinya

b. L = 0 penuh kiri

c. R < n - 1 bisa insert kanan

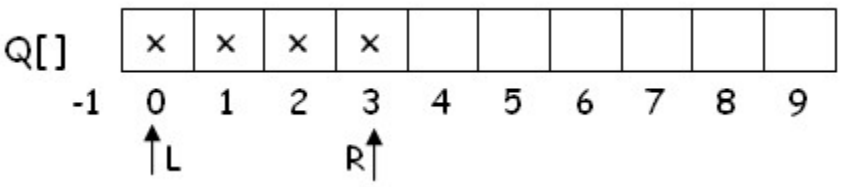


3) Misal masuk lagi 3 pengantri dari kanan

a. L < R + 1 ada isinya

b. L = 0 penuh kiri

c. R < n - 1 bisa insert kanan

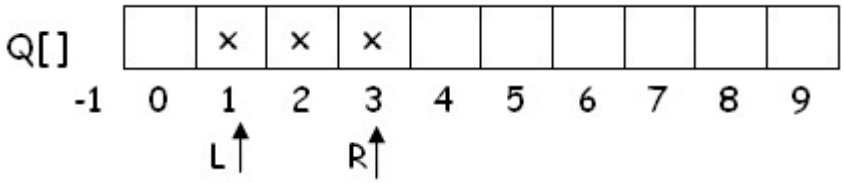


4) Misal keluar 1 pengantri dari kiri

a. L < R + 1 ada isinya

b. L > 0 bisa insert kiri

c. R < n - 1 bisa insert kanan

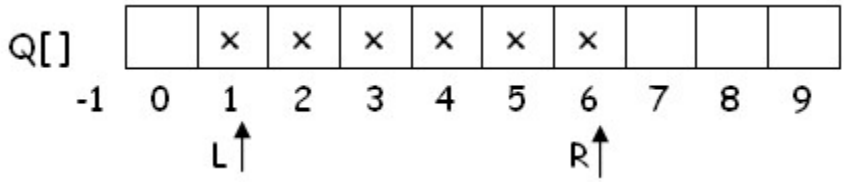


5) Misal masuk lagi 3 pengantri dari kanan

a. L < R + 1 ada isinya

b. L > 0 bisa insert kiri

c. R < n – 1 bisa insert kanan



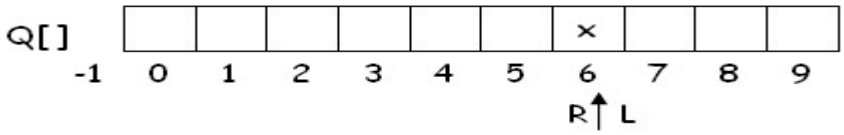
6) Misal keluar 5 pengantri melalui kiri

a. L < R + 1 ada isinya

b. L > 0 bisa insert kiri

c. R < n – 1 bisa insert kanan

d. L = R hanya 1 pengantri

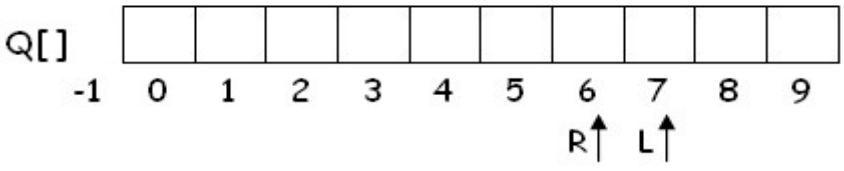


7) Misal keluar 1 pengantri melalui kiri

a. L = R + 1 ada isinya

b. L > 0 bisa insert kiri

c. R < n - 1 bisa insert kanan

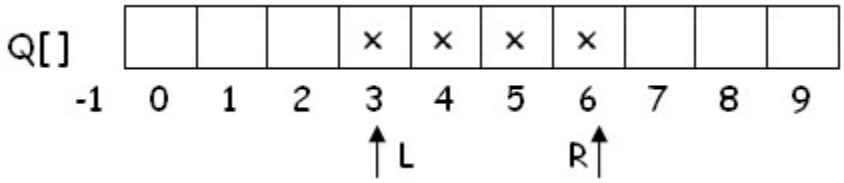


8) Misal masuk 4 pengantri dari kiri

a. L < R + 1 ada isinya

b. L > 0 bisa insert kiri

c. R < n – 1 bisa insert kanan

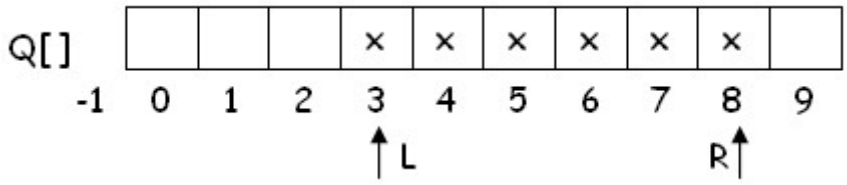


9) Misalkan masuk lagi 2 pengantri dari kanan

a. L < R + 1 ada isinya

b. L > 0 bisa insert kanan

c. R < n – 1 bisa insert kanan

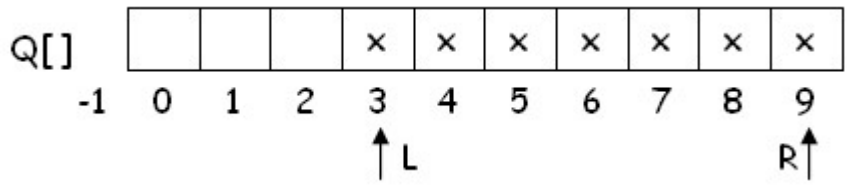


10) Misal masuk lagi 1 pengantri dari kanan

a. L < R + 1 ada isinya

b. L > 0 bisa insert kiri

c. R = n – 1 penuh kanan

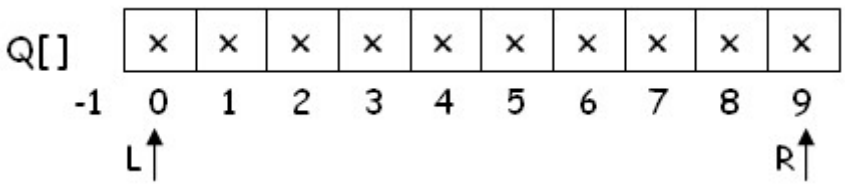


11) Misal masuk lagi 3 pengantri dari kiri

a. L < R + 1 ada isinya

b. L = 0 penuh kiri

c. R = n - 1 penuh kanan

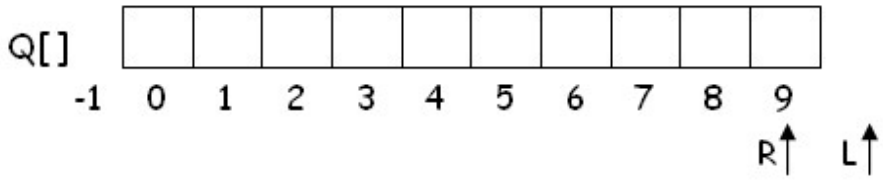


12) Misal seluruh pengantri keluar dari kiri

a. L = R + 1 antrian kosong

b. L > 0 bisa insert kiri

c. R = n – 1 penuh kanan



4. Kondisi antrian:

	Kondisi	Ciri
a	Kosong	L = R + 1
b	Penuh kiri	L = 0
c	Penuh kanan	R = n -1
d	Bisa diisi dari kiri	L > 0
e	Bisi diisi dari kanan	R < n – 1
f	Ada isinya	L < R + 1

5. Fungsi lengkap

a) INSERT KIRI lengkap

```
void INSERT_KIRI(void)
{
        if(L > 0)
        {
                Q[--L] = x;
        }
        else cout<<"Antrian penuh kiri";
}
```

b) INSERT KANAN lengkap

```
void INSERT_KANAN(void)
{
        if(R < n - 1)
        {
                Q[++R] = x;
        }
        else cout<<"Antrian penuh kanan";
}
```

c) DELETE KIRI lengkap

```
void DELETE_KIRI(void)
{
        if(L < R + 1)
        { x = Q[L++];
        }
        else cout<<"Antrian kosong";
}
```

d) DELETE KANAN lengkap

```
void DELETE_KANAN(void)
{
        if(L < R + 1)
        { x = Q[R--];
        }
        else cout<<"Antrian kosong";
}
```

Listing Program deklarasi dari Double Ended Queue

```
#define MAX 5

//Deklarasi Double Ended Queue
class DQueue
{
   private:
      int front   ;
      int rear    ;
      int count   ;
      int ele[MAX]    ;

   public:
      DQueue();
      int  isFull();
      int  isEmpty();
      void insertDQueueAtRear(int item);
      int  deleteDQueueAtFont(int *item);
      void insertDQueueAtFront(int item);
      int  deleteDQueueAtRear(int *item);
};
```

Jenis-jenis DeQueue

1. Input Restricted DeQueue
2. Output Dibatasi DeQueue

Dalam Input Restricted DeQueue, penyisipan hanya dapat dilakukan dari REAR, tetapi penghapusan dapat dilakukan dari FRONT dan REAR.

Listing Program dari Implementasi DeQueue dengan semua operasi Queue di atas

```
#include <iostream>
using namespace std;
#define MAX 5

//Deklarasi  Double Ended Queue
class DQueue
{
   private:
      int front   ;
      int rear    ;
      int count   ;
      int ele[MAX]    ;
```

```
    public:
        DQueue();
        int  isFull();
        int  isEmpty();
        void insertDQueueAtRear(int item);
        int  deleteDQueueAtFont(int *item);
        void insertDQueueAtFront(int item);
        int  deleteDQueueAtRear(int *item);
};

//Inisialisasi  Double Ended Queue
DQueue:: DQueue()
{
    front =  0;
    rear  = -1;
    count =  0;
}

// Untuk memeriksa DQueue sudah penuh atau tidak
int DQueue:: isFull()
{
    int full=0;

    if( count == MAX )
        full = 1;

    return full;
}

// Untuk memeriksa DQueue kosong atau tidak
int DQueue:: isEmpty()
{
    int empty=0;

    if( count == 0 )
        empty = 1;

    return empty;
}

// Masukkan item ke DQueue
void DQueue:: insertDQueueAtRear(int item)
{
    if( isFull() )
```

```
    {
        cout<<"\nQueue Overflow";
        return;
    }

    rear= (rear+1) % MAX;

    ele[rear] = item;
    count++;

    cout<<"\nDimasukkan di Belakang FRONT : "<< front<<", REAR : "<< rear;
    cout<<"\nItem dimasukkan : "<< item << endl;
}

//Delete item from DQueue
int DQueue:: deleteDQueueAtFont(int *item)
{
    if( isEmpty() )
    {
        cout<<"\nQueue Underflow";
        return -1;
    }
    *item = ele[front];
    front = (front+1) % MAX;

    cout<<"\nSetelah Hapus Di Depan FRONT : "<< front<<", REAR : "<< rear;
    count--;
    return 0;
}

//Untuk memasukkan item ke dalam Dequeue di Front.
void DQueue:: insertDQueueAtFront(int item)
{
    if( isFull() )
    {
        cout<<"\nQueue Overflow";
        return;
    }

    if ( front == 0)
        front = MAX - 1;
    else
        front = front - 1;
    ele[front] = item;
```

```
    count++;
    cout<<"\nSetelah Sisipkan Di Depan FRONT :"<< front<<", REAR : "<< rear;
    cout<<"\nItem yang disisipkan : "<< item<< endl;
}

//Untuk menghapus item dari Dequeue di Rear.
int DQueue:: deleteDQueueAtRear(int *item)
{
    if( isEmpty() )
    {
        cout<<"\nQueue Underflow";
        return -1;
    }

    *item = ele[rear];

    if(rear == 0)
        rear = MAX - 1;
    else
        rear = rear - 1;

    cout<<"\nSetelah Hapus Dibelakang FRONT : "<< front<<", REAR : "<< rear;
    count--;
    return 0;
}

int main()
{
    int  item;
    DQueue q = DQueue();

    q.insertDQueueAtRear(10);
    q.insertDQueueAtRear(20);
    q.insertDQueueAtRear(30);
    q.insertDQueueAtFront(40);
    q.insertDQueueAtFront(50);
    q.insertDQueueAtFront(60);

    if( q.deleteDQueueAtFont(&item) == 0 )
    {
        cout<<"\nItem dihapus:"<< item<< endl;
    }

    if( q.deleteDQueueAtFont(&item) == 0 )
    {
```

```
        cout<<"\nItem dihapus:"<< item<< endl;
    }

    if( q.deleteDQueueAtFont(&item) == 0 )
    {
        cout<<"\nItem dihapus:"<< item<< endl;
    }

    if( q.deleteDQueueAtRear(&item) == 0 )
    {
        cout<<"\nItem dihapus:"<< item<< endl;
    }

    if( q.deleteDQueueAtRear(&item) == 0 )
    {
        cout<<"\nItem dihapus:"<< item<< endl;
    }

    if( q.deleteDQueueAtRear(&item) == 0 )
    {
        cout<<"\nItem dihapus:"<< item<< endl;
    }
    cout<<"\n";
    return 0;
}
```

Hasil output dari program diatas yaitu

```
Dimasukkan di Belakang FRONT : 0, REAR : 0
Item dimasukkan : 10

Dimasukkan di Belakang FRONT : 0, REAR : 1
Item dimasukkan : 20

Dimasukkan di Belakang FRONT : 0, REAR : 2
Item dimasukkan : 30

Setelah Sisipkan Di Depan FRONT :4, REAR : 2
Item yang disisipkan : 40

Setelah Sisipkan Di Depan FRONT :3, REAR : 2
Item yang disisipkan : 50

Queue Overflow
Setelah Hapus Di Depan FRONT : 4, REAR : 2
```

```
Item dihapus:50

Setelah Hapus Di Depan FRONT : 0, REAR : 2
Item dihapus:40

Setelah Hapus Di Depan FRONT : 1, REAR : 2
Item dihapus:10

Setelah Hapus Dibelakang FRONT : 1, REAR : 1
Item dihapus:30

Setelah Hapus Dibelakang FRONT : 1, REAR : 0
Item dihapus:20

Queue Underflow
```

C. SOAL LATIHAN/TUGAS

Latihan 8

1. Tulis algoritma dasar double ended queue untuk kondisi:
    a. Inisialisasi
    b. Insert sebuah record dari kanan
    c. Insert sebuah record dari kiri
    d. Delete sebuah record dari kanan
    e. Delete sebuah record dari kiri
2. Sebutkan ciri bahwa double ended queue pada:
    a. Kosong tidak ada isinya
    b. Penuh kanan, tak bisa diisi dari kanan
    c. Penuh kiri, tidak bisa diisi dari kiri
    d. Penuh total, tidak bisa diisi dari kiri maupun dari kanan
    e. Hanya diisi 10 pengantri
3. Tulis algoritma yang lengkap untuk mengisi antrian dari kanan record per record sampai antrian penuh kanan, atau tidak bisa diisi lagi dari kanan.
4. Tulis algoritma lengkap untuk mendelete isi antrian dari kanan record per record sampai antrian kosong
5. Tulis algoritma yang lengkap untuk mengisi antrian dari kanan record per record sebanyak 10 record selama antrian belum penuh kanan. Apabila antrian sudah penuh kanan, walaupun belum mengisi record, proses pengisian dihentikan.

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 9
LINEAR SINGLY LINKED LIST

A. TUJUAN PEMBELAJARAN

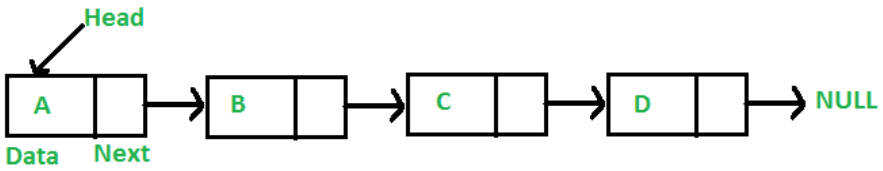
Pada bab ini akan dijelaskan mengenai Mampu memahami dan merepresentasikan konsep algoritma *Linier Singly Linked List* :

1. Mampu memahami dan merepresentasikan konsep algoritma *Linier Singly Linked List*

B. URAIAN MATERI

Single Linked List

Seperti array, Linked list adalah struktur data linier. Tidak seperti array, elemen linked list tidak disimpan di lokasi yang berdekatan; elemen-elemennya dihubungkan menggunakan pointer.



Gambar 9.1 Linked List

Mengapa Linked List?

Array dapat digunakan untuk menyimpan data linier dengan tipe yang sama, tetapi array memiliki batasan berikut.

1) Ukuran array sudah tetap: Jadi kita harus mengetahui batas atas jumlah elemen terlebih dahulu. Selain itu, secara umum, memori yang dialokasikan sama dengan batas atas terlepas dari penggunaannya.
2) Memasukkan elemen baru ke dalam array elemen mahal karena ruangan harus dibuat untuk elemen baru dan untuk membuat ruangan, elemen yang ada harus digeser.
   Misalnya, dalam sistem, jika kita memelihara daftar ID yang diurutkan dalam id array [].

   id [] = [1000, 1010, 1050, 2000, 2040].

Dan jika kita ingin memasukkan ID baru 1005, maka untuk mempertahankan urutan yang diurutkan, kita harus memindahkan semua elemen setelah 1000 (tidak termasuk 1000). Penghapusan juga mahal dengan larik sampai kecuali beberapa teknik khusus digunakan. Misalnya, untuk menghapus 1010 di id [], semua setelah 1010 harus dipindahkan.

Keuntungan dibandingkan array

1) Ukuran dinamis
2) Kemudahan penyisipan / penghapusan

Kekurangan:

1) Akses acak tidak diperbolehkan. Kita harus mengakses elemen secara berurutan mulai dari node pertama. Jadi kita tidak dapat melakukan pencarian biner dengan daftar tertaut secara efisien dengan implementasi defaultnya.
2) Ruang memori tambahan untuk penunjuk diperlukan dengan setiap elemen dari daftar.
3) Tidak ramah cache. Karena elemen array adalah lokasi yang bersebelahan, ada lokalitas referensi yang tidak ada dalam kasus daftar tertaut.

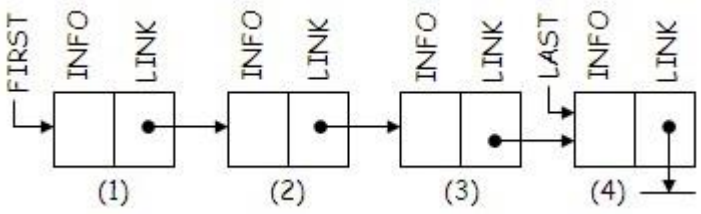
Link List: sejumlah obyek yang dilink/dihubungkan satu dengan lainnya.

Obyek : gabungan bebrapaelemen data yg dijadikan satu kelompok/struktur/record

Untuk menghubungkan antar obyek perlu variabel tipe pointer yg merupakan salah satu variabel dalam struktur obyek.

Linear Singly Linked List : Link list lurus dengan pointer tunggal

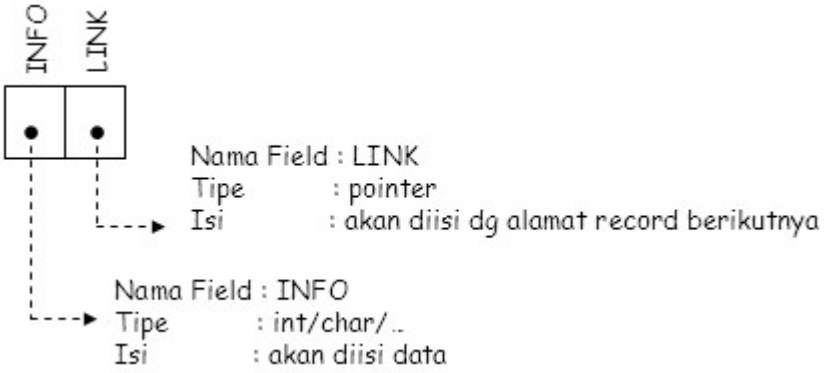
1. Ilustrasi



- Ada 4 simpul : 1, 2, 3, 4
- Setiap simpul terdiri dari 2 elemen/field, yaitu :
  - INFO : bertipe integer o LINK :

  bertipe pointer Simpul no.1 :
  - Field INFO berisi 10 o Field LINK

  berisi alamat simpul no. 2
- Simpul no.1 ditunjuk oleh pointer FIRST
- Simpul no.4 ditunjuk oleh pointer LAST Ilustrasi sebuah simpul :



Untuk mempersiapkan sebuah linked list maka harus dideklarasikan sbb :

```
struct SIMPUL{ int INFO; struct
        SIMPUL *LINK;
};
SIMPUL *P,*Q,*FIRST,*LAST;
```

2. Proses
    a. Inisialisasi : persiapan pembuatan linked list
    b. Membuat simpul awal
    c. Insert simpul kedalam linked list
    d. Delete simpul dari linked list

1) Inisialisasi

```
FIRST = NULL;
LAST = NULL;
```

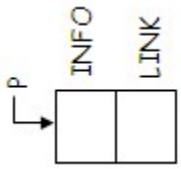
2) Pembuatan simpul

Instruksi untuk membuat sebuah simpul :

```
P=(SIMPUL*) malloc(sizeof(SIMPUL));
```

Akan terbentuk sebuah simpul yang alamatnya tersimpan dalam pointer P.

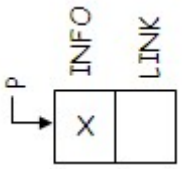
Ilustrasi :



Fungsi untuk membuat simpul :

```
void BUAT_SIMPUL(int X)
{
P=(SIMPUL*)
malloc(sizeof(SIMPUL)); if(P!=NULL)
{
P->INFO=X;
}
else cout<<"Pembuatan simpul
gagal";
}
```

Ilustrasi :



Contoh :

```
#include<iostream.h>
#include<stdlib.h> struct SIMPUL{    int
INFO;  struct SIMPUL *LINK;
        };
        SIMPUL *P,*FIRST,*LAST; void
BUAT_SIMPUL(int); void main(void)
        {
            int x;
            cout<<"Masukan Data : ";cin>>x;
            BUAT_SIMPUL(x);
            cout<<"Data : "<<P->INFO<<endl;
}
void BUAT_SIMPUL(int x)
{
          P=(SIMPUL *)malloc(sizeof(SIMPUL));
          if(P!=NULL)
                    P->INFO=x;
          else
                     cout<<"Pembuatan Simpul Gagal"<<endl;
}
```

3. Pembuatan simpul awal

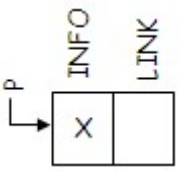
Menjadikan sebuah simpul menjadi simpul awal dari sebuah linked list. Simpul awal ditunjuk oleh pointer FIRST.

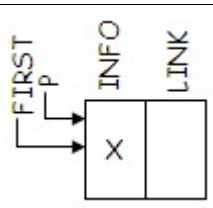
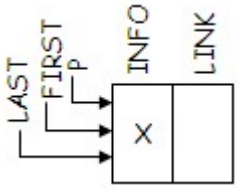
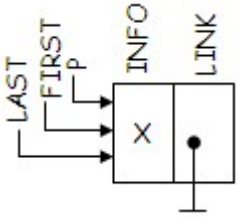
Fungsi :

```
Void AWAL(void)
{
    if(FIRST==NULL)
    {
        FIRST=P;
        LAST=P;
        P->LINK=NULL;
    }
    else cout<<"Linked List sudah ada""<<endl;
}
```

Ilustrasi :

Sudah dibuat simpul sbb:



FIRST=P	
LAST=P atau LAST=FIRST	
P->LINK=NULL atau FIRST->LINK=NULL atau LAST->LINK=NULL	

3) Insert Kanan

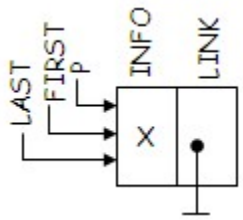
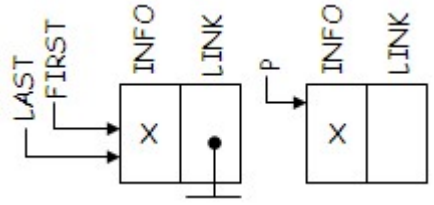
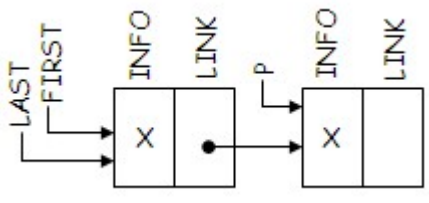
Menyisipkan sebuah simpul baru pada ujung kanan linked list.

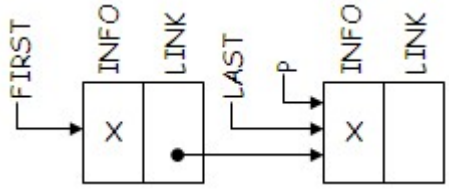
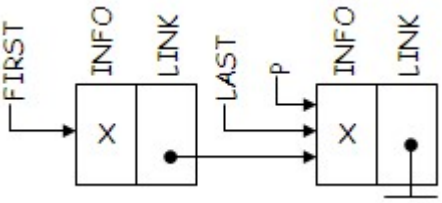
Proses :

- sudah ada linked list
- buat simpul baru
- sisipkan simpul baru tsb diujung kanan linked list Fungsi :

```
void INSERT_KANAN(void)
{
    if(LAST!=NULL)
    {
        LAST->LINK=P;
        LAST=P;
        P->LINK=NULL;
    }
    else cout<<"Linked List belum
        ada";
}
```

Ilustrasi :

Sudah ada linked list	
Buat simpul baru P=(SIMPUL *)malloc(sizeof(SIMPUL));	
LAST->LINK=P atau FIRST->LINK=P	

LAST=P atau LAST=FIRST->LINK	
P->LINK=NULL atau LAST->LINK=NULL atau FIRST->LINK- >LINK=NULL	

4) Insert Kiri

Menyisipkan sebuah simpul baru pada ujung kiri linked list.

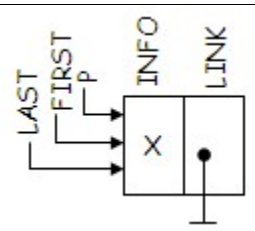
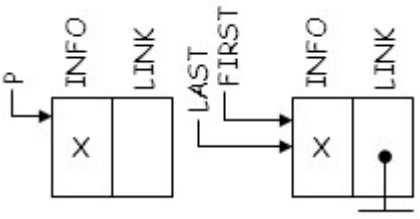
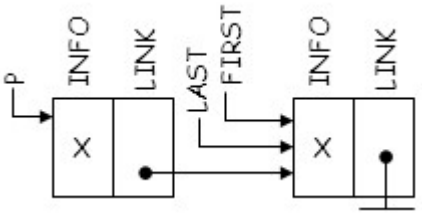
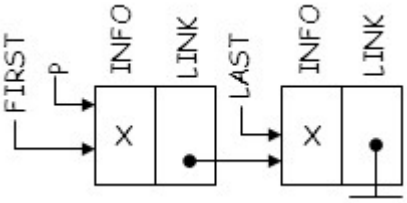
Proses :

- sudah ada linked list
- buat simpul baru
- sisipkan simpul baru tsb diujung kiri linked list

Fungsi :

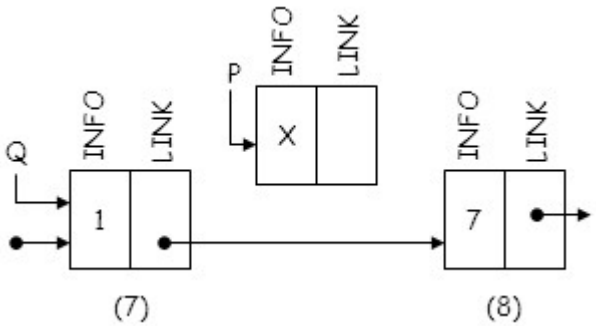
```
void INSERT_KIRI(void)
{
    if(FIRST!=NULL)
    {
        P->LINK=FIRST;
        FIRST=P;
    }
    else cout<<"Linked List belum ada";
    }
```

Ilustrasi:

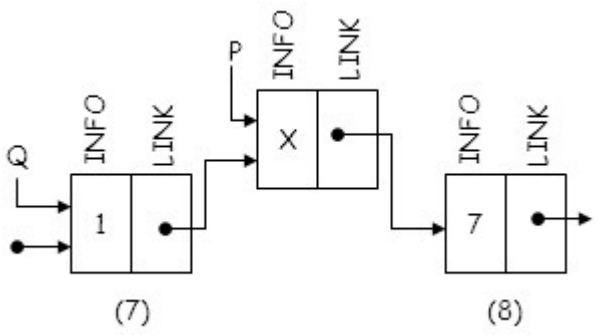
Sudah ada linked list	
Buat simpul baru P=(SIMPUL *)malloc(sizeof(SIMPUL));	
P->LINK=FIRST atau P->LINK=LAST	
FIRST=P	

II.6. Insert Tengah

Menyisipkan sebuah simpul antara dua buah simpul pada linked list.



setelah diinsert menjadi :



Syarat : simpul no.7 harus sudah ditunjuk oleh pointer Q, caranya :

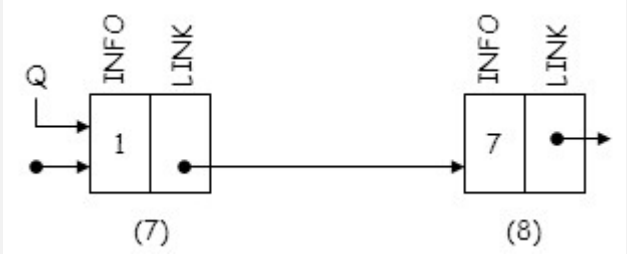
```
Q=FIRST;
For(i=1;i<=6;i++)
Q=Q->LINK;
```

Fungsi :

```
Void INSERT_TENGAH(void)
{
    P->LINK=Q->LINK;
    Q->LINK=P;
}
```

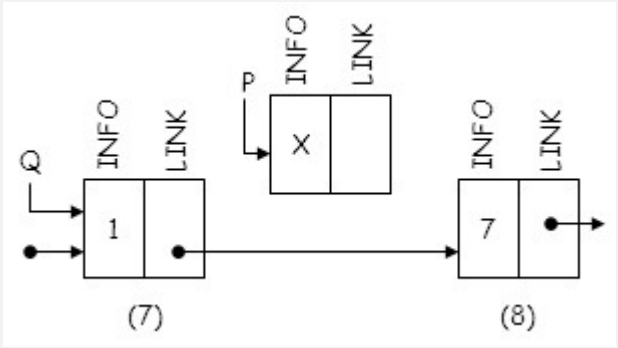
Ilustrasi :

Sudah ada linked list

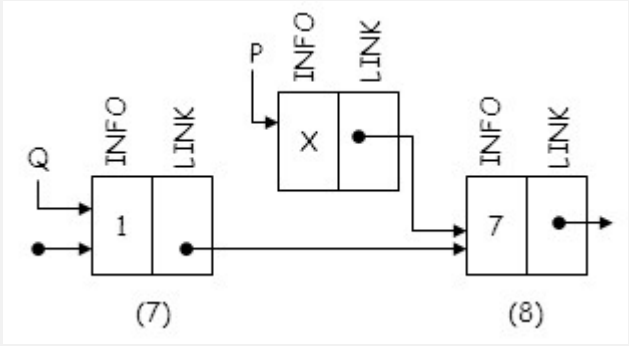


Buat simpul baru

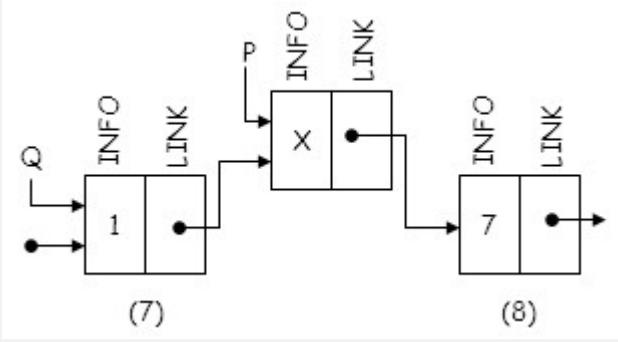
P=(SIMPUL *)malloc(sizeof(SIMPUL));



P->LINK=Q->LINK



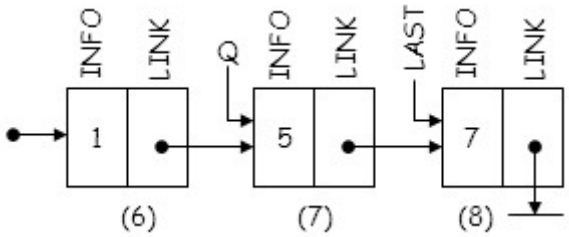
Q->LINK=P



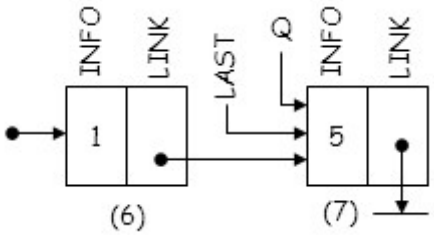
5) Delete Kanan/Akhir

Menghapus simpul yang ada pada linked list paling akhir/kanan.

Ilustrasi : sudah ada sebuah linked list



akan dihapus simpul terakhir menjadi :



Syarat agar simpul no.8 dapat dihapus adalah simpulno.7 sudah ditunjuk oleh pointer Q.

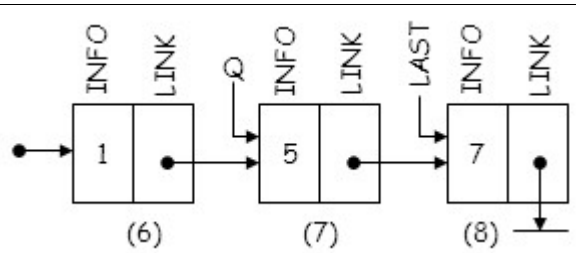
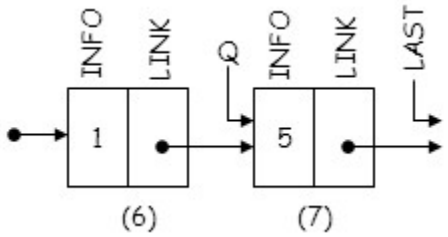
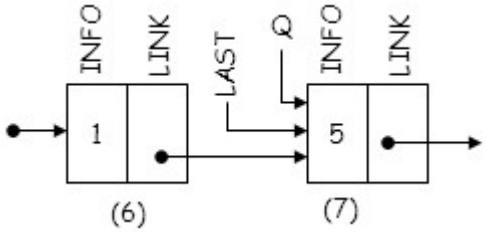
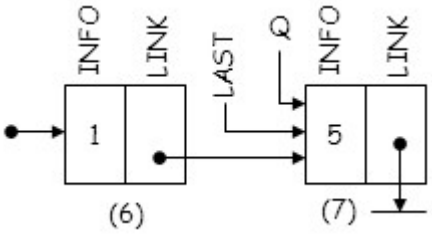
Caranya :

```
Q = FIRST;
while(Q ->LINK != LAST)
        Q = Q -> LINK;
```

Fungsi :

```
void DELETE_KANAN(void)
{
    free(LAST);
    LAST = Q;
     LAST -> LINK = NULL;
}
```

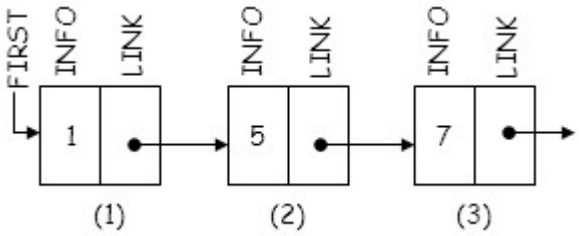
Ilustrasi :

Sudah ada linked list	
free(LAST)	
LAST = Q	
LAST -> LINK = NULL	

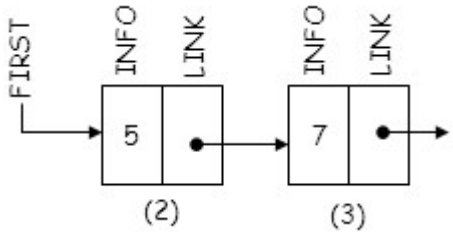
II.8. Delete Kiri/Awal

Menghapus simpul yang ada pada linked list paling awal/kiri.

Ilustrasi : sudah ada sebuah linked list



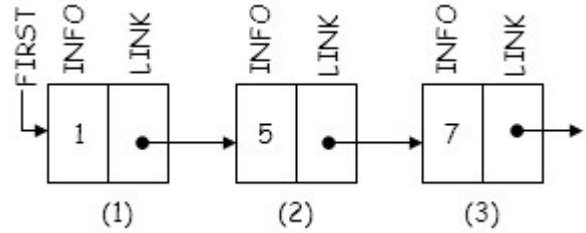
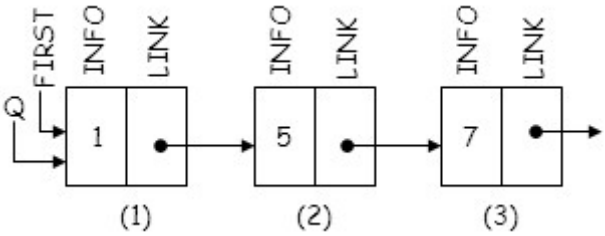
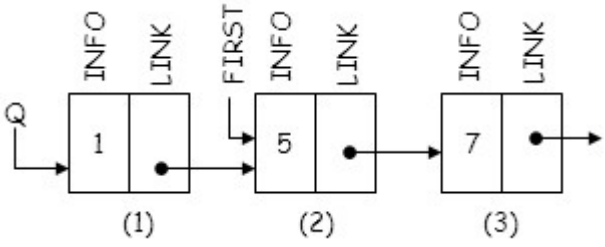
akan dihapus simpul awal menjadi :

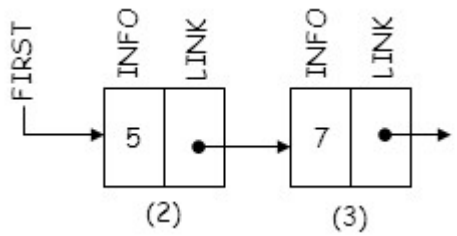


Fungsi :

```
void DELETE_KIRI(void)
{
    Q = FIRST;
    FIRST = Q -> LINK;
    free(Q);
}
```

Ilustrasi :

Sudah ada linked list	
Q = FIRST	
FIRST = Q -> LINK	

free(Q)	

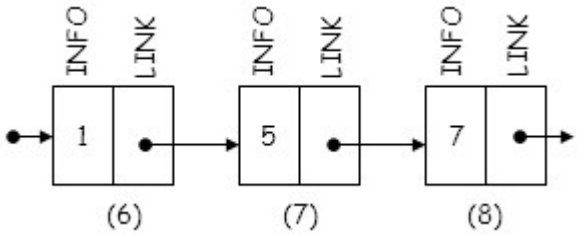
Tuliskan cara lain!

Tip : Tempatkan Q pada simpul kedua, hapus simpul pertama, pindahkan FIRST ke simpul kedua.

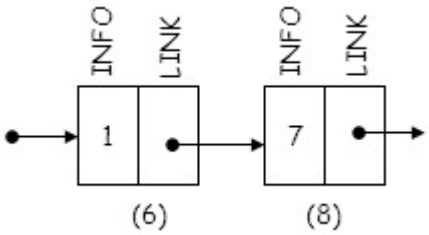
6) Delete Tengah

Menghapus simpul yang ada diantara dua simpul lain.

Ilustrasi : sudah ada linked list



simpul no.7 akan dihapus sehingga menjadi :



Syarat agar simpul no.7 bisa dihapus maka simpul no.6 harus sudah ditunjukoleh Q.

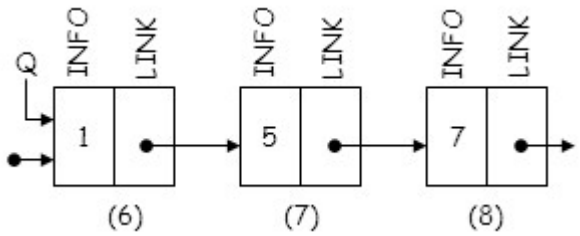
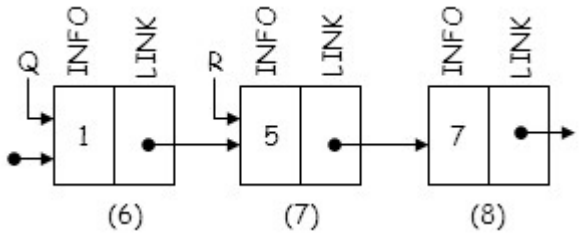
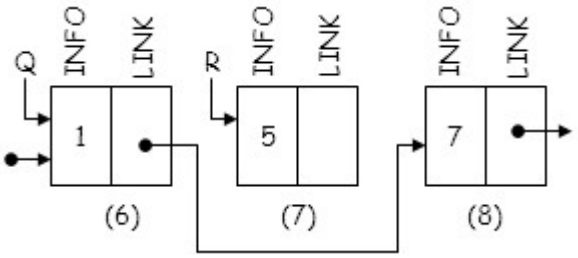
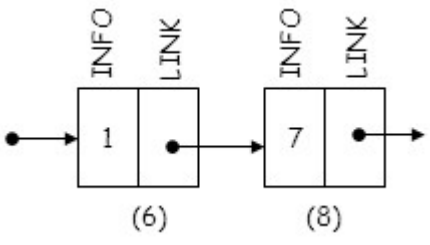
Caranya :

```
Q = FIRST;
For(I = 1; I <= 5; I++)
Q = Q -> LINK;
```

Fungsi :

```
void DELETE_TENGAH(void)
{
    R = Q -> LINK;
    Q -> LINK = R -> LINK;
    free(R);
}
```

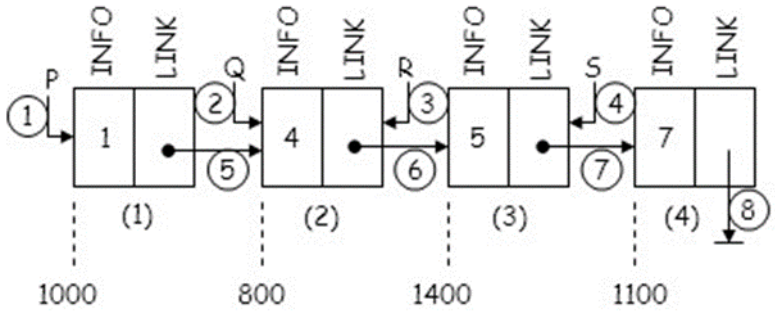
Ilustrasi :

Sudah ada linked list	
R = Q -> LINK	
Q -> LINK = R -> LINK	
free(R)	

C. SOAL LATIHAN/TUGAS

Latihan 9.

1. Perhatikan penggalan linked List seperti gambar berikut:

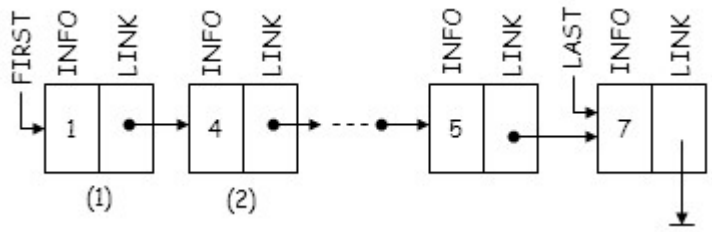


   a. Sebutkan nama dan isi tiap-tiap pointer
   b. Sebutkan pointer-pointer yang bernilai sama

   a) Sebutkan TRUE atau FALSE kondisi pada intruksi tiap pernyataan dibawah ini :

      i. if(P->LINK==R)
      ii. if(Q->LINK==R->LINK)
      iii. if(Q->LINK->LINK==S->LINK)
      iv. if(Q==R)
      v. if(Q-LINK==R)
      vi. if(R->LINK->INFO==5)
      vii. if(Q->INFO==4)

2. Sudah ada linked list yang diilustrasikan seperti gambar dibwah ini, Simpul pertama ditunjuk oleh pointer FIRST, dan simpul terakhir ditunjuk oleh pointer LAST. Jumlah simpul tepatnya tidak diketahui, tapi dipastikan lebih dari 10 simpul. LINK dari simpul terakhir nilainya = NULL



   Susun algoritma untuk menempatkan Pointer Q sehingga menunjuk:

   a. Simpul no (1)

b. Simpul no (7)
c. Simpul akhir
d. Simpul dengan nilai INFO = 50
e. Simpul yang letaknya satu simpul didepan (disebelah kiri) simpul dengan nilai INFO = 50

3. Sudah ada Linked List seperti no 2. Diatas.
   a. Menghitung dan mencetak jumlah simpul
   b. Menghitung dan mencetak TotalINFO (25+12+…….+27,14)
   c. Mencetak semua nilai INFO ke layar
   d. Mencetak jumlah simpul yang nilai INFOnya = 50

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 10
APLIKASI SINGLE LINKED LIST PADA STACK

A. TUJUAN PEMBELAJARAN

Pada bab ini akan dijelaskan tentang mengilustrasikan dan mengaplikasikan *Linked List* pada *Stack*

1. Mampu mengilustrasikan dan mengaplikasikan *Linked List* pada *Stack*

B. URAIAN MATERI

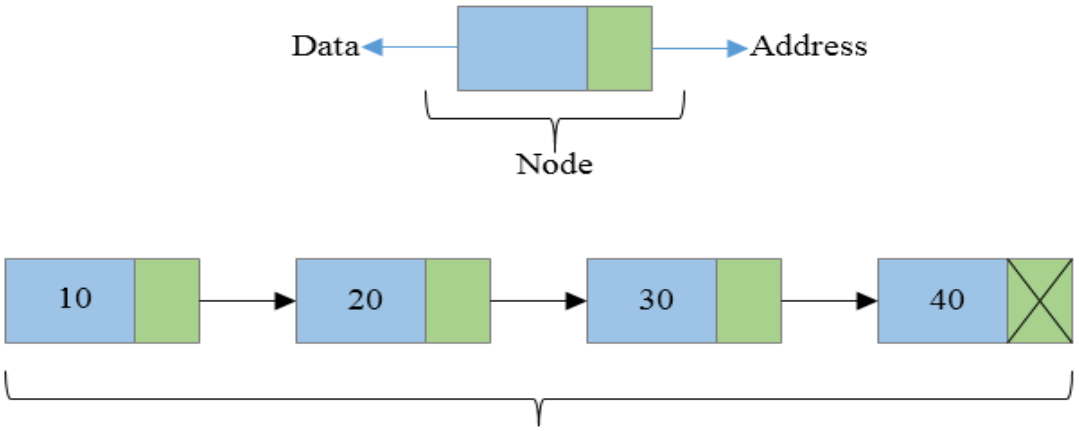
Pada pertemuan sebelumnya kita telah berkenalan dengan single linked list, yang merupakan sebuah struktur data linier dimana elemen-elemennya dihubungkan menggunakan pointer, dan tidak harus tersimpan pada suatu lokasi memori yang berdekatan.

Single linked list adalah jenis daftar tertaut dasar. Single linked list adalah kumpulan node yang dihubungkan bersama secara berurutan di mana setiap node dari daftar tertaut tunggal berisi bidang data dan bidang alamat yang berisi referensi dari node berikutnya. Daftar tertaut tunggal dapat berisi beberapa bidang data tetapi harus berisi setidaknya satu bidang alamat yang menunjuk ke simpul berikutnya yang terhubung.

Pada pertemuan ini kita akan mempelajari tentang bagaimana penerapan konsep single linked list pada stack / tumpukan. Operasi single linked list kali ini dilakukan berdasarkan pada karakteristik dari operasi Stack, yaitu LIFO (*Last In First Out*). Di pengaplikasian single linked list pada stack, kita akan mempelajari bagaimana sebuah linked list kita gunakan untuk menyimpan informasi dalam bentuk node dan tentunya dengan mengikuti aturan dari stack / tumpukan.

Pada hakikatnya, stack dapat direpresentasikan dengan menggunakan array, dengan syarat banyaknya elemen maksimal suatu stack tidak melebihi batas maksimum banyaknya array, karena jika banyaknya elemen stack melebihi batas maksimum dari array maka akan terjadi kondisi yang dinamakan *overflow*. Berangkat dari masalah ini, serta dengan memahami fakta bahwasannya jumlah elemen stack sangat bervariasi atau dinamis, maka bentuk penyajian stack dengan menggunakan array menjadi kurang tepat. Alasannya jelas, karena banyaknya elemen pada array pada dasarnya adalah statis.

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.



Gambar 10.1 Singly Linked List

Terdapat beberapa operasi yang digunakan dalam pengaplikasian single linked list pada stack. Beberapa diantaranya yang perlu kita kenali adalah :

1) Pendeklarasian *stack*

Proses pendeklarasian stack menggunakan single linked list, hanya memerlukan suatu pointer yang menunjuk ke data terakhir. Setiap elemen linked list mempunyai 2 field yaitu datanya serta tautan yang merujuk ke posisi terakhir sebelum proses push.

```
Struck SIMPUL{
        int INFO;
        struck SIMPUL *LINK;
};
Typedef struct Node Simpul;
Simpul *P,*Q,*TOP,*DASAR;
```

2) Inisialisasi

Proses inisialisasi untuk pengaplikasian single linked list pada stack adalah dengan cara mengisi nilai pointer stack dengan Null.

```
TOP = NULL;
DASAR = NULL;
```

3) Operasi IsEmpty

Operasi IsEmpty pada stack yang diaplikasikan dengan single linked list adalah dengan melakukan pemeriksaan nilai dari pointer stack, apakah ia bernilai NULL. Jika stack bernilai NULL maka itu tandanya stack sedang dalam keadaan kosong dan akan me-return-kan nilai 1. Dan bila pointer stack tidak bernilai NULL, maka itu tandanya stack tidak kosong (ada isinya) sehingga operasi tersebut akan me-return-kan nilai false (0).

4) Operasi IsFull

Dikarenakan sifat dari linked list yang dinamis, maka proses pengecekan isFull adalah dalam kontek memeriksa ketersediaan memori, apakah masih bisa digunakan untuk alokasi sebuah elemen stack. Apabila alokasi dapat dilaksanakan, maka artinya memori belum penuh dan masih memiliki ruang yang cukup untuk memuat dan proses push dapat dilakukan. Namun bila alokasi memori gagal dilaksanakan, maka artinya memori telah penuh dan tidak bisa lagi menambah atau memuat elemen stack baru.

Berikut adalah contoh operasi pemeriksaan stack IsEmpty dan IsFull pada C++ :

```
bool isEmpty() {
        return Tumpukan.top == -1;
}
bool isFull() {
        return Tumpukan.top == MAX-1;
}
```

5) Operasi Push

Operasi push pada stack yang direpresentasikan oleh single linked list adalah sama dengan proses pembuatan simpul baru pada operasi linked list. Dimana langkah-langkahnya adalah sebagai berikut :

a. Memeriksa terlebih dahulu ketersediaan memory, apakah memori penuh (isfull). Yang mana jika bernilai false/0 (tidak penuh) maka proses push akan dijalankan dan jika pemeriksaan ini bernilai true/1 (stack penuh), maka proses push akan dibatalkan.
b. Proses push dilakukan dengan cara mengalokasikan suatu elemen linked list (disebut variable baru), kemudian melakukan pemeriksaan jika stack dalam

keadaan kosong maka pointer yang menunjuk ke awal stack diisi dengan pointer baru, dan jika dengan menambah field top dengan 1, kemudian elemen pada posisi top diisi dengan elemen data baru. Namun bila pointer penunjuk stack sudah menunjuk ke suatu data, maka sambungkan field pointer link (penunjuk ke data sebelumnya) dari pointer baru ke pointer penunjuk posisi akhir stack (top) dan kemudian pindahkah pointer penunjuk posisi akhir stack ke pointer baru.

```
void PUSH(void)
{
if(DASAR!=NULL)
{
P->LINK=TOP;
TOP=P;
}
else cout<<”Stack belum ada”;
}
```

6) Operasi Pop

Adapun langkah-langkah operasi pop pada stack yang direpresentasikan oleh single linked list adalah sama dengan proses hapus awal pada operasi single linked list. Langkah-langkah prosesnya adalah sebagai berikut :

a. Melakukan pemeriksaan terlebih dahulu apakah stack kosong (isempty), jika kosong maka proses pop tidak bisa dilaksanakan. Jika stack tidak kosong maka proses pop akan dijalankan.

b. Proses pop dilakukan dengan mengambil elemen yang ditunjuk oleh pointer stack, lalu simpan dalam variable data. Kemudian dilanjutkan dengan pembuatan variable pointer bantu yang diisi dengan pointer penunjuk stack yang selanjutnya akan dihapus dari memori. Kemudian pointer penunjuk stack dipindahkan ke posisi yang ditunjuk oleh field pointer bawah dari variable bantu.

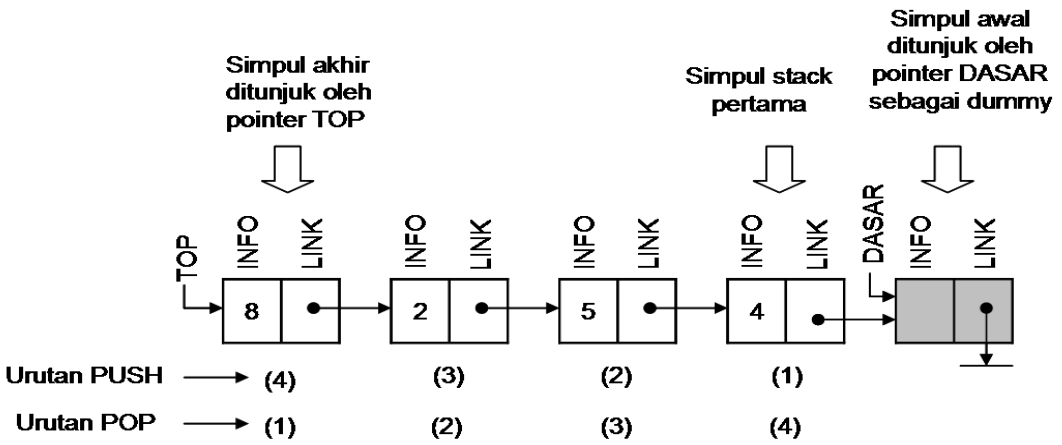
```
Int POP(void)
{ int X;
    If(TOP!=DASAR)
{
            X =TOP->INFO; Q=TOP->LINK; free(TOP); TOP=Q;
            return(X);
}
            else cout<<”Stack Kosong”;
}
```

Di bawah ini terdapat dua contoh cara dalam penyajian stack / tumpukan dengan menggunakan single linked list, yaitu ketika stack menggunakan simpul head / kepala, dimana simpul awalnya ditunjuk oleh pointer dasar sebagai simpul palsu / dummy.

Cara kedua adalah ketika stack tidak menggunakan simpul head / kepala, atau tidak menggunakan simpul awal palsu / dummy.

Dan berikut adalah pengaplikasian keduanya dengan single linked list beserta ilustrasinya :

1. **Ilustrasi Stack Menggunakan Simpul Head**



PUSH : Selalu Insert Kiri

POP : Selalu Delete Kiri

Intruksi /Fungsi-fungsi yang diperlukan :

1) Deklarasi struktur simpul dan pointer yang diperlukan

```
struck SIMPUL{
        int INFO;
        struck SIMPUL *LINK;
};
Typedef struct Node Simpul;
Simpul *P,*Q,*TOP,*DASAR;
```

2) Inisialisasi stack

```
TOP = NULL;
```

```
DASAR = NULL:
```

3) Fungsi pembuatan simpul baru

```
void BUAT_SIMPUL(int X)
{P=(SIMPUL*)
malloc(sizeof(SIMPUL));
    if(P!=NULL)
      {
        P->INFO=X;
      }
    else
      { cout<<"Membuat simpul gagal";
        exit(1);
      }
```

4) Fungsi pembuatan simpul Head

```
void BUAT_HEAD(void)
{ if(DASAR==NULL)
     { DASAR=P;
       TOP=DASAR;
       DASAR->LINK=NULL;
       DASAR->INFO=0; }
  else
       cout<<"Head sudah ada";
}
```

5) Fungsi PUSH/Insert Kiri

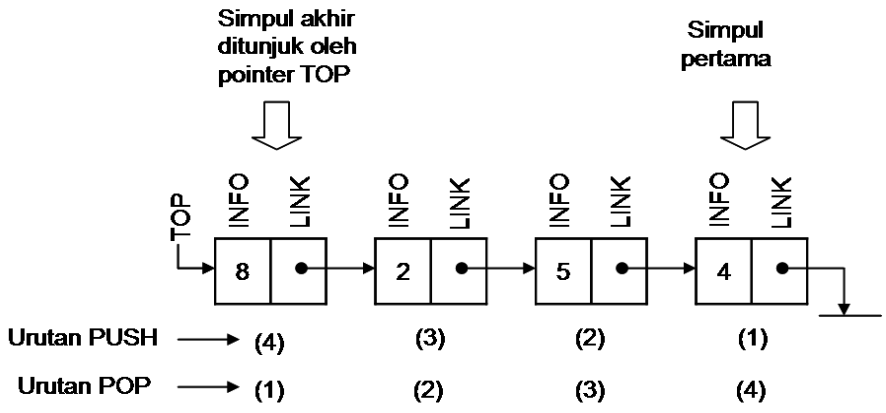
```
void PUSH(void)
{
if(DASAR!=NULL)
{
    P->LINK=TOP;
    TOP=P;
}
```

```
        else cout<<"Stack belum ada";
    }
```

6) Fungsi POP/Delete Kiri

```
int POP(void)
{ int X;
if(TOP!=DASAR)
{
    X=TOP->INFO;  Q=TOP->LINK;
    free(TOP); TOP=Q;
    return(X);
}
    else cout<<"Stack kosong";
}
```

2. **Ilustrasi Untuk Stack Tanpa Menggunakan Simpul Head**



PUSH : selalu insert kiri

POP : selalu delete kiri

Jika TOP->LINK=NULL berarti isi stack tinggal satu simpul (simpul pertama) dan bila simpul ini dihapus (POP) maka TOP dibuat sama dengan NULL.

Intruksi/Fungsi-fungsi yang diperlukan :

1) Deklarasi struktur simpul dan pointer yang diperlukan

```
struck SIMPUL{
        int INFO;
        struck SIMPUL *LINK;
    };
Typedef struct Node Simpul;
Simpul *P,*Q,*TOP;
```

2) Inisialisasi stack

```
TOP = NULL;
DASAR = NULL:
```

3) Fungsi pembuat Simpul Baru

```
void BUAT_SIMPUL(int X)
{P=(SIMPUL*)malloc(sizeof(SIMPUL));
if(P!=NULL)
    {
        P->INFO=X;
    }
else
   { cout<<"Membuat simpul gagal";
     exit(1);
   }
```

4) Fungsi PUSH (Insert Kiri atau Buat Awal)

```
void PUSh(void)
{
  if(TOP==NULL)
      {
        TOP=P;
        TOP->LINK=NULL;
      }
  else
      {
        P->LINK=TOP;
        TOP=P;
      }
}
```

5) Fungsi POP (Delete Kiri)

```
int POP(void)
{ int X;
  if(TOP!=NULL)
    {
      X=TOP->INFO;
      Q=TOP->LINK;
      free(TOP);
      TOP=Q;
      return(X); }
  else
       cout<<"Stack kosong";
}
```

Listing Program untuk mengimplementasikan stack menggunakan singly Linked list

// Program C ++ untuk mengimplementasikan tumpukan
//Menggunakan singly linked list
#include <bits/stdc++.h>
using namespace std;

```
// Deklarasikan node linked list

struct Node
{
    int data;
    struct Node* link;
};

struct Node* top;

// Fungsi utilitas untuk menambahkan elemen
// data dalam tumpukan dimasukkan di awal
void push(int data)
{

    // Buat node baru dan alokasikan memori
    struct Node* temp;
    temp = new Node();

    // Periksa apakah tumpukan (heap) sudah penuh.
    // Kemudian memasukkan elemen yang akan menyebabkan stack
overflow
    if (!temp)
    {
        cout << "\nHeap Overflow";
        exit(1);
    }

    // Inisialisasi data ke data field
    temp->data = data;

    // letakkan top pointer referensi ke dalam temp link
    temp->link = top;

    // Jadikan sebagai bagian atas Stack
    top = temp;
}

// Fungsi utilitas untuk memeriksa apakah tumpukan kosong atau tidak
int isEmpty()
{
    return top == NULL;
}

// Fungsi utilitas untuk mengembalikan elemen teratas dalam stack
int peek()
{
```

```
    // Periksa tumpukan kosong
    if (!isEmpty())
        return top->data;
    else
        exit(1);
}

// Fungsi utilitas untuk pop top
// elemen dari tumpukan
void pop()
{
    struct Node* temp;

    // Cek untuk stack underflow
    if (top == NULL)
    {
        cout << "\nStack Underflow" << endl;
        exit(1);
    }
    else
    {

        // Top assign into temp
        temp = top;

        // Assign second node to top
        top = top->link;

        // Hilangkan pembatas
        // pertama dan kedua
        temp->link = NULL;

        // Lepaskan memori node teratas
        free(temp);
    }
}

// Berfungsi untuk mencetak semua file elemen stack
void display()
{
    struct Node* temp;

    // cek untuk stack underflow
    if (top == NULL)
    {
        cout << "\nStack Underflow";
        exit(1);
```

```
    }
    else
    {
        temp = top;
        while (temp != NULL)
        {

            // Cetak data node
            cout << temp->data << "-> ";

            // Assign temp link to temp
            temp = temp->link;
        }
    }
}

// Driver Code
int main()
{

    // Push elemen ke dalam  stack
    push(11);
    push(22);
    push(33);
    push(44);

    // Menampilkan elemen stack
    display();

    // Mencetak elemen  stack paling atas
    cout << "\nElemen Top adalah "
         << peek() << endl;

    // Hapus elemen teratas stack
    pop();
    pop();

    // Menampilkan elemen stack
    display();

    // Cetak elemen stack paling atas
    cout << "\nElemen Top adalah"
         << peek() << endl;

    return 0;
}
```

Hasil output dari program diatas

```
44-> 33-> 22-> 11->
Elemen Top adalah 44
22-> 11->
Elemen Top adalah22
```

C. SOAL LATIHAN/TUGAS

Latihan 10

1. Jelaskan pengertian dari konsep LIFO
2. Jelaskan pengertian dari konsep FIFO
3. Jelaskan penerapan aplikasi STACK dengan Array

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 11
APLIKASI SINGLE LINKED LIST PADA QUEUE

A. TUJUAN PEMBELAJARAN

Pada bab ini akan dijelaskan tentang bagaimana mengilustrasikan dan mengaplikasikan *Single Linked List* pada *Queue*

1. Mampu mengilustrasikan dan mengaplikasikan *Single Linked List* pada *Queue*

B. URAIAN MATERI

Pada pertemuan 6 kita telah mempelajari seputar queue (antrian), yang mana merupakan suatu struktur data linier yang memiliki suatu aturan pengurutan tertentu pada operasi yang dilakukan. Aturan pengurutan pada queue tentunya berbeda dengan aturan pengurutan pada stack.

Jika pada stack (tumpukan) ada istilah *LIFO (Last In First Out)* maka pada queue (antrian) ada aturan yang digunakan yaitu *FIFO (First In First Out)*.

Pada pertemuan ini kita akan mempelajari tentang bagaimana pengimplementasian konsep single linked list pada *queue* (antrian). Operasi single linked list kali ini dilakukan berdasarkan pada karakteristik dari operasi queue, yaitu *FIFO* (*Last In First Out*).

Dalam struktur data Queue, ada dua penunjuk, Front (depan) dan Rear (belakang). Titik depan item pertama antrian dan titik belakang untuk item terakhir.

Penerapan queue / antrian secara dinamis dapat dilakukan menggunakan tipe data pointer. Di bawah ini adalah beberapa operasi yang digunakan secara umum pada queue :

1) Buat_queue / create()

   Operasi ini digunakan sebagai pendeklarasian queue yang kosong atau menginisialisasi queue yang kosong, yaitu dengan membuat front (head) dan rear (tail) = NULL.

2) isEmpty()

   Operasi ini digunakan untuk memeriksa kondisi dari queue yang mana melihat kondisi apakah sudah penuh atau belum dengan memeriksa nilai rear, jika rear = NULL maka empty. Kita tidak bisa memeriksa front

dikarenakan ia merupakan tanda utama sebagai kepala queue yang sejatinya tidak akan berubah-ubah. Penambahan elemen antrian di belakang dengan menggunakan nilai rear akan mengakibatkan pergerakan pada queue.

3) isFull()

   Operasi ini digunakan untuk melakukan pengecekan kondisi queue apakah kondisi queue sudah penuh atau belum, dengan mengecek nilai rearnya.

4) Clear()

   Operasi ini digunakan untuk menghapus elemen-elemen queue.

5) Tampil_queue()

   Operasi ini digunakan untuk memperlihatkan nilai-nilai elemen pada queue dengan looping dari front sampai dengan rear.

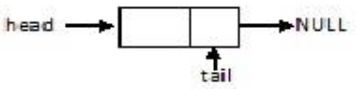
6) enQueue()

   Operasi ini digunakan untuk menambahkan ketika elemen masuk ke dalam antrian, yang mana penambahan tersebut terjadi pada elemen yang posisinya paling akhir.

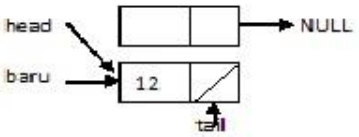
Ilustrasi Enqueue:

a. Tahap 1 (penambahan node ke 1: 12)

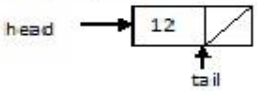
head = tail = NULL

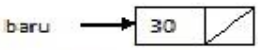


head = baru; tail = baru (misal 12)

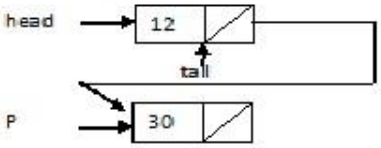


b. Tahap 2 (penambahan node ke 2: 30)

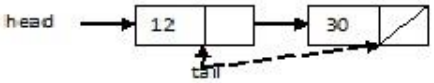


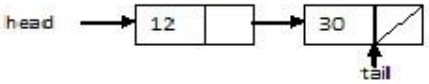


tail → next = baru

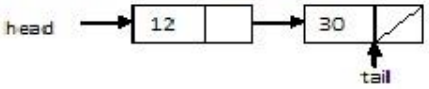


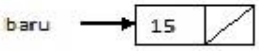
tail = baru



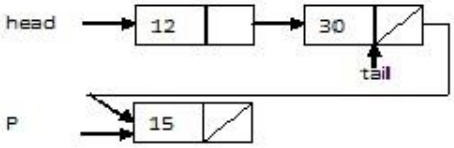


c. Tahap 3 (penambahan node ke 3: 15)

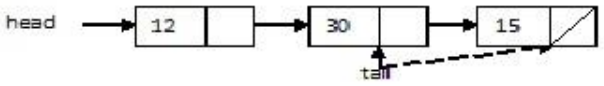


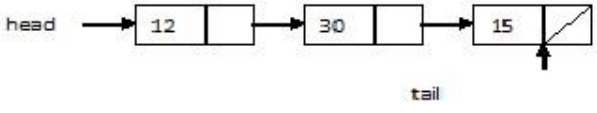


tail → next = baru



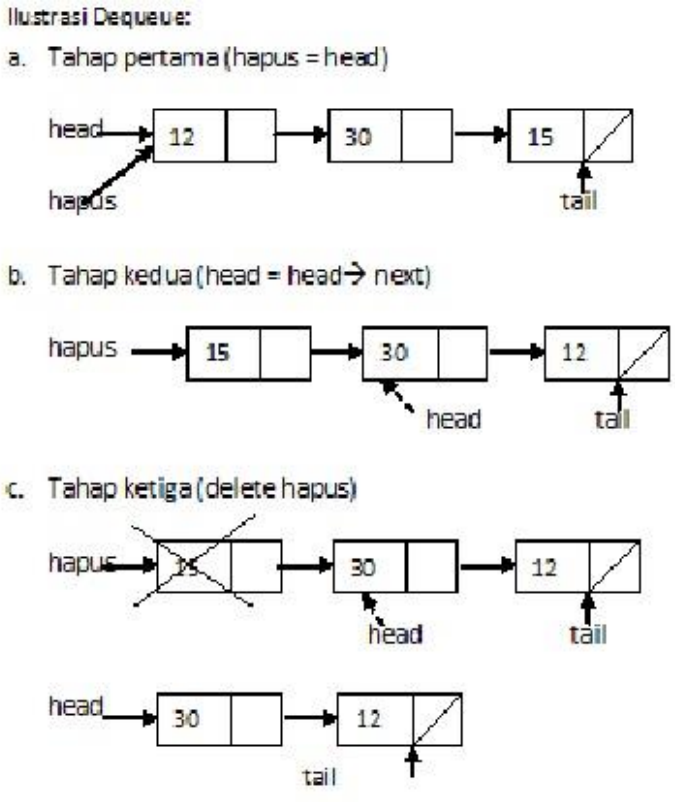
tail = baru





7) deQueue()

Pada operasi ini digunakan untuk mengeluarkan elemen pada posisi paling depan atau yang terdepan di antrian, prosesnya dengan mengurangi counter rear (tail) serta memajukan semua elemen antrian ke depan.



8) Rear

Operasi ini digunakan untuk melihat elemen paling belakang dari antrian.

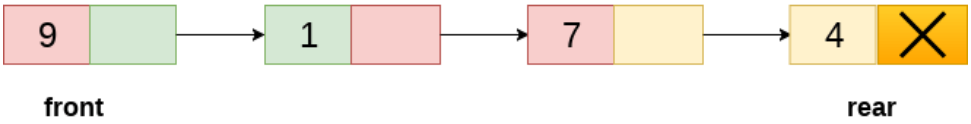
9) Front

Operasi ini digunakan untuk melihat elemen terdepan dari antrian.

Elemen-elemen dari queue disimpan pada posisi simpul yang dibuat pada memori dinamis. Pada setiap simpulnya memiliki 2 bagian, satu bagiannya merupakan field data yang berisi elemen dari data antrian, dan satu bagian lagi berisi field link yang berisi pointer yang merujuk ke simpul berikutnya dalam antrian.

Adapun struktur dari suatu queue mampu memuat field sebagai berikut :

- depan : pointer ke node pertama dalam antrian
- belakang : pointer ke node terakhir dalam antrian



Pada setiap proses penambahan elemen pada pada queue, elemen tersebut tentunya akan disimpan pada posisi terakhir. Setelah proses penambahan elemen tersebut selesai, maka variable yang tadinya merupakan bagian akhir, akan menunjuk ke data baru tersebut.

Hal tersebut mengharuskan adanya 2 kondisi yang wajib diperiksa yaitu kondisi penambahan pada queue yang masih kosong dan kondisi penambahan queue yang telah memiliki elemen.

Fungsi di atas tentunya sama dengan fungsi penambahan simpul di belakang, pada linked list. Yang mana langkah-langkahnya adalah sebagai berikut :

1) Pembuatan simpul baru yang kemudian diisi dengan informasi baru.
2) Proses penghubungan simpul paling belakang dengan simpul baru.
3) Pengarahan pointer yang ada di simpul belakang menunjuk ke simpul baru.

Proses mengambil data pada queue kurang lebih sama dengan proses penghapusan elemen pertama pada linked list, yang menyebabkan posisi awal queue akan bergeser ke elemen queue berikutnya.

Ada tiga kondisi yang harus diperhatikan yaitu :

- Kondisi queue kosong
- Kondisi queue hanya memiliki satu data antrian
- Kondisi queue yang memiliki data lebih dari 1 elemen.

Adapun langkah-langkah yang perlu dilakukan adalah :

1) Mengarahkan simpul bantu ke simpul depan.
2) Mengarahkan simpul depan ke simpul berikutnya.
3) Menghapus simpul bantu dari queue.

Perlu diketahui bahwasannya queue-pun dapat direpresentasikan dengan menggunakan array. Terdapat beberapa perbandingan antara implementasi queue menggunakan array dengan implementasi queue menggunakan linked list.

Implementasi queue menggunakan array cenderung lebih sederhana, yang mana ukuran memori yang akan digunakan harus ditentukan terlebih dahulu saat objek queue dideklarasikan.

Untuk kelemahan dari implementasi queue menggunakan array yaitu terjadinya inefisiensi / penggunaan memori yang sia-sia tatkala jumlah data yang digunakan ternyata lebih sedikit dari alokasi memori yang telah ditentukan sebelumnya. Dan kelemahan kedua adalah ketidakfleksibelan untuk menambahkan data melebihi ukuran array yang telah dideklarasikan sebelumnya.

Adapun untuk pengimplementasian queue dengan menggunakan linked list, pengalokasian memori berjalan secara dinamis, tanpa melakukan segala sesuatu terkait alokasi memori, serta menggunakan dua buah pointer front dan rear untuk menandai posisi depan dan belakang dari queue.

Jadi dapat disimpulkan, untuk queue yang berukuran besar yang dalam hal ini tidak diketahui secara pasti jumlah maksimalnya, sangat direkomendasikan untuk menggunakan linked list. Terlebih lagi untuk perangkat yang memiliki ukuran memori yang terbatas seperti small handheld device, linked list memiliki performa yang lebih baik.

Listing program pengaplikasian single linked list

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *next;
};
struct Node* head = NULL;
void insert(int new_data) {
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = head;
   head = new_node;
}
void display() {
   struct Node* ptr;
   ptr = head;
   while (ptr != NULL) {
      cout<< ptr->data <<" ";
```

```
      ptr = ptr->next;
   }
}
int main() {
   insert(3);
   insert(1);
   insert(7);
   insert(2);
   insert(9);
   cout<<" linked list adalah: ";
   display();
   return 0;
}
```

Output dari hasil pemograman diatas

```
linked list adalah: 9 2 7 1 3
```

Dalam program di atas, struktur Node membentuk node linked list. Ini berisi data dan penunjuk ke node linked list berikutnya. Ini diberikan sebagai berikut.

```
struct Node {
   int data;
   struct Node *next;
};
```

Fungsi insert () menyisipkan data ke awal linked list. Ini membuat new_node dan menyisipkan nomor di bidang data new_node. Kemudian new_node mengarah ke head. Akhirnya head adalah new_node yaitu linked list dimulai dari sana. Ini diberikan di bawah ini.

```
void insert(int new_data) {
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = head;
   head = new_node;
}
```

Fungsi display () menampilkan seluruh daftar tertaut. Pointer pertama ke kepala. Kemudian terus menerus diteruskan ke node berikutnya sampai semua nilai data dari node tersebut dicetak. Ini diberikan di bawah ini.

```
void display() {
   struct Node* ptr;
   ptr = head;
   while (ptr != NULL) {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
   }
}
```

Dalam fungsi main (), berbagai nilai pertama dimasukkan ke dalam linked list dengan memanggil insert (). Kemudian linked list ditampilkan. Ini diberikan di bawah ini.

```
int main() {
   insert(3);
   insert(1);
   insert(7);
   insert(2);
   insert(9);
   cout<<"Linked list adalah: ";
   display();
   return 0;
}
```

C. SOAL LATIHAN/TUGAS

Latihan 11
1. Buat suatu program animasi Stack yang menggunakan Linked List dalam mengelola data mahasiswa dengan struktur mahasiswa sbb : NAMA, NIM, GENDER, NILAI STRUKTUR DATA. Program dibuat dalam bentuk menu dengan pilihan : INSERT DATA, HAPUS DATA, CETAK DATA, EXIT 2. Buat suatu program animasi Stack yang menggunakan Linked List tanpa Head untuk mengelola data mahasiswa dengan struktur mahasiswa sbb : NAMA, NIM, GENDER, NILAI STRUKTUR DATA. Program dibuat dalam bentuk menu dengan pilihan : INSERT DATA, HAPUS DATA, CETAK DATA, EXIT.

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 12
LINEAR DOUBLE LINKED LIST

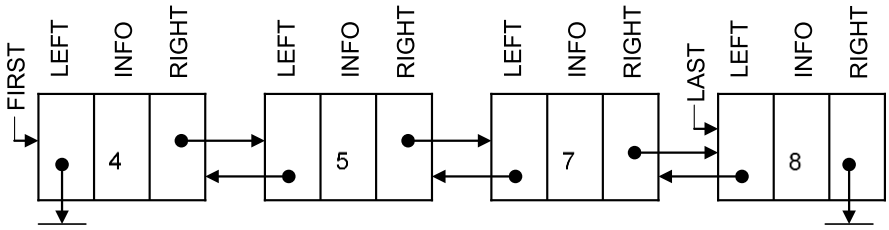
A. TUJUAN PEMBELAJARAN

Materi pada bab ini akan dijelaskan mengenai dalam memahami, menginisialisasi dan membuat algoritma Insert Pada *Linear Double Linked List*.

1. Mampu dalam memahami, menginisialisasi dan membuat algoritma Insert Pada *Linear Double Linked List*

B. URAIAN MATERI

Linier Doubly Linked list (DLL) berisi penunjuk tambahan, biasanya disebut penunjuk sebelumnya, bersama dengan penunjuk dan data berikutnya yang ada dalam linke list dengan nya. variasi lain pada doubly linkedlist : doubly linked list (jangan disamakan dengan doubly ended list) apa keuntungan dari doubly linked list? Masalah potensial dengan double linked list adalah sulit untuk menelusuri ke bagian akhir sepanjang list pada sebuah pernyataan.



Gambar 11.1 Linier Doubly Linked list

Masalah utamannya adalaha Langkah mudah ke link berikutnya, tetapi tidak ada cara yang sesuai untuk pergi ke link sebelumnya. Bergantung pada aplikasinya, batasan ini dapat menimbulkan masalah.

Misalnya, bayangkan editor teks dimana linkedlist digunakan untuk menyimpan teks. Baris teks halaman dilayar disimpan sebagai objek string yang disematkan di link. Ketika pengguna editor menggunakan kursor ke bawah pada layar, program melangkah ke link berikutnya untuk memanipulasi atau menampilkan baris baru. Tetapi apa yang terjadi jika pengguna menggunakan kursor keatas? Dalam linkedlist biasa, kamu perlu mengembalikan current (atau

yang setara dengan) awal list dan kemudian turun ke bawah lagi ke link current yang baru. Ini tidak terlalu efisien. Anda ingin membuat satu langkah keatas.

Doubly linkedlist menyediakan kemampuan ini, hal ini memungkinkan anda untuk menelusuri mundur dan maju melalui list. Rahasianya adalah setiap link memiliki dua referensi ke link lain, bukan hanya satu.. yang pertama adalah ke link berikutnya dalam linkedlist biasa. Yang kedua adalah ke link sebelumnya.

Berikut adalah keuntungan / kerugian dari double linked list atas daftar terkait nya.

Keuntungan daripada daftar yang ditautkan dengan nya yang tertaut

1. DLL dapat dilalui dalam arah maju dan mundur.
2. Operasi hapus di DLL lebih efisien jika pointer ke node yang akan dihapus diberikan.
3. Kita dapat dengan cepat memasukkan node baru sebelum node tertentu.

Dalam daftar tertaut, untuk menghapus node, pointer ke node sebelumnya diperlukan. Untuk mendapatkan node sebelumnya, kadang-kadang daftar dilalui. Dalam DLL, kita bisa mendapatkan node sebelumnya menggunakan pointer sebelumnya.

Kekurangan atas daftar yang terkait dengan nya

1. Setiap node DLL memerlukan ruang ekstra untuk pointer sebelumnya. Hal ini dimungkinkan untuk menerapkan DLL dengan pointer tunggal meskipun.
2. Semua operasi memerlukan pointer tambahan sebelumnya untuk dipertahankan. Misalnya, dalam penyisipan, kita perlu memodifikasi pointer sebelumnya bersama dengan petunjuk berikutnya. Misalnya dalam fungsi berikut untuk penyisipan pada posisi yang berbeda, kita perlu 1 atau 2 langkah tambahan untuk mengatur pointer sebelumnya.

1. **Traversal**

Dua metode tampilan menunjukkan travelsal dalam doubly linkedlist metode displayforward () sama dengan metode displaylist() yang telah kita lihat di linkedlist biasa. Metode displaybackward() serupa tetapi dimulai dari yang terakhir. List di element melanjutkan ke list awal menuju elemen lain.

Kebetulan, beberapa orang berpandangan bahwa, dikarenakan anda bisa pergi dengan mudah menggunakan doubly linkedlist, tidak ada arah yang disukai dan oleh karena itu istilah seperti previous dan next tidak sesuai. Jika mau, anda dapat mengganti symbol left dan right.

2. **Insertion**

Kami telah memasukkan beberapa rutinitas di dalam metode doubly linked list insertfirst() menyisipkan di awal daftar, insertlast() menyisipkan diakhir dan insertafter() menyisipkan mengikuti elemen dengan kunci yang ditentukan.

Kecuali jika listnya kosong insertfirst() mengubah field sebelumnya di link pertama yang lama untuk menunjuk ke link baru dan megubah field di link baru menjadi link utama. Akhirnya ini diatur terlebih dahulu untuk menunjuk ke link baru.

Metode insertlast() merupakan proses yang sama yang diterapkan ke akhir list itu adalah bayangan dari mirror insertfirst().

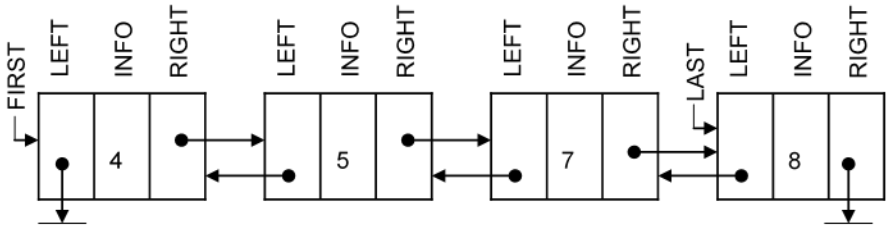
Metode insertafter() menyisipkan link baru setelah link dengan nilai kunci yang ditentukan. Ini sedikit lebih rumit karena 4 sambungan harus dibuat. Pertama link dengan nilai kunci yang ditentukan harus ditemukan. Prosedur ini ditangani dengan cara yang sama seperti find() dalam program linklist.java (listing 5.2). kemudian dengan asumsi kita berada diakhir daftar 2 koneksi harus dibuat antara link baru dan link berikutnya, dan 2 lagi antara link saat ini dan yang baru.

3. **Deletion**

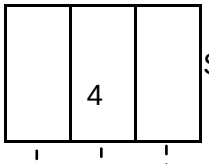
Ada tiga rutinitas penghapusan : deletefirst(), deletelast() dan deletekey(). Dua yang pertama cukup mudah didalam deletekey() kunci yang dihapus adalah yang terbaru. Dengan asumsi link yang akan dihapus bukan yang pertama atau yang terakhir dalam list. Field berikutnya dari currentprevious(link setelah yang dihapus), dan kolom current.next sebelumnya diatur ke currentprevious. Ini memutuskan link current dari list.

Metode penghapusan dan metode insertafter() mengasumsikan bahwa list tidak kosong. Meskipun kesederhanaan kami tidak menampilkan di main(),isEmpty() harus digunakan untuk memverifikasi bahwa ada suatu dalam list sebelum mencoba penyisipan dan penghapusan seperti itu

4. **Ilustrasi**



Ilustrasi Simpul :



Simpul dengan 3 elemen atau

Nama Field : RIGHT

Tipe : pointer

Isi : akan diisi alamat record berikutnya

Nama Field :

INFO Tipe :

integer

Isi : akan diisi

Nama Field : LEFT

Tipe : pointer

Isi : akan diisi alamat record sebelumnya

Deklarasi :

```
struct SIMPUL{ int INFO; struct SIMPUL *LEFT; struct SIMPUL *RIGHT;
};
SIMPUL *P,*FIRST,*LAST;
```

II. Proses

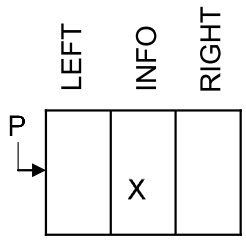
II.1.Inisialisasi

```
FIRST = NULL;
LAST =NULL;
```

II.2.Pembuatan Simpul

```
void BUAT_SIMPUL(int X)
{
P=(SIMPUL*) malloc(sizeof(SIMPUL)); if(P!=NULL)
{
P->INFO=X;
}
else
{
cout<<"Pembuatan simpul gagal"; exit(1);
    }
}
```

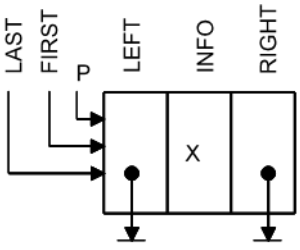
Akan terbentuk sebuah simpul:



II.3.Pembuatan simpul awal

```
void AWAL(void)
{
          FIRST=P;
          LAST=P;
          P->LEFT=NULL;
            P->RIGHT=NULL;
```

Ilustrasi :



```
}
```

II.4.Insert Kanan

Proses yang dilakukan dengan cara menyisipkan sebuah simpul baru pada ujung kanan linked list.

Proses :

- sudah ada linked list
- buat simpul baru
- sisipkan simpul baru tsb diujung kanan linked list

Fungsi :

```
void INSERT_KANAN(void)
{
        LAST->RIGHT=P;
        P->LEFT=LAST;
        LAST=P;
        P->RIGHT=NULL;
}
```

II.5.Insert Kiri

Proses yang dilakukan dengan cara menyisipkan sebuah simpul baru pada ujung kiri linked list.

Proses :

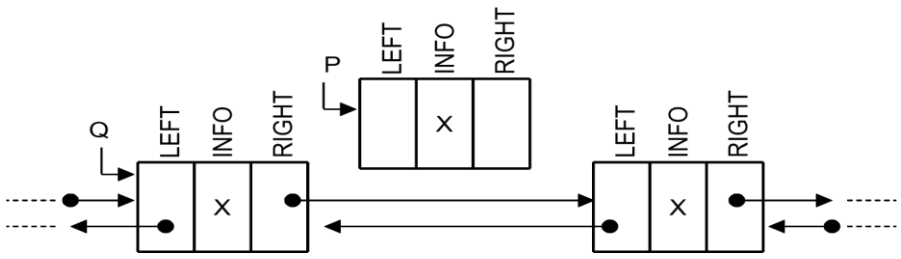
- sudah ada linked list
- buat simpul baru
- sisipkan simpul baru tsb diujung kiri linked list

Fungsi :

```
void INSERT_KIRI(void)
{
        P->RIGHT=FIRST;
      FIRST->LEFT=P;
       FIRST=P;
      P->LEFT=NULL;
}
```

II.6.Insert Tengah

Proses yang dilakukan dengan cara menyisipkan sebuah simpul antara dua buah simpul pada linked list. Misal akan menyisipkan sebuah simpul antara simpul no 7 dan simpul no 8.



(7) (8)

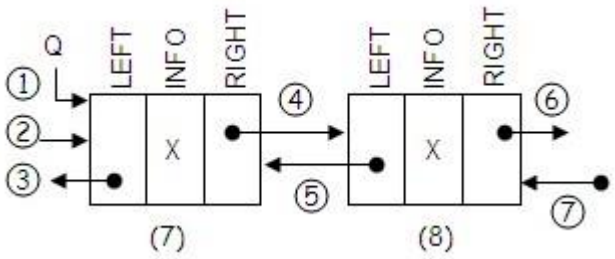
Fungsi :

```
void INSERT_TENGAH(void)
{
        P->RIGHT=Q->RIGHT;
        P->LEFT=Q;
            Q->RIGHT->LEFT=P;
        Q->RIGHT=P;
}
```

C. SOAL LATIHAN/TUGAS

Latihan.12

1. Sudah ada Linked List sbb:



   a. Sebutkan nama-nama pointer sesuai dengan nomornya
   b. Sebutkan pointer yang nilainya sam

2. Akan dibuat Linked List dalam mengelola data mahasiswa dengan komponen struktur terdiri dariNIM, NAMA,NILAI. Data tersusun naik berdasarkan NILAI.
   a. Buatlah program untuk mengisikan data baru
   b. Buatlah program untuk menampilkan data dengan NILAI sama dengan 90
   c. Buatlah program untuk menampilkan seluruh data
   d. Buatlah program untuk menghapus data dengan NIM sama dengan 2007140022.
   e. Buatlah program untuk menghitung nilai rata-rata kelas

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Sjukani, M. (2012). *Struktur Data (Algoritma dan Struktur Data 2), Edisi 5.* Jakarta: Mitra Wacana Media.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 13

LINEAR DOUBLE LINKED LIST (LANJUTAN)

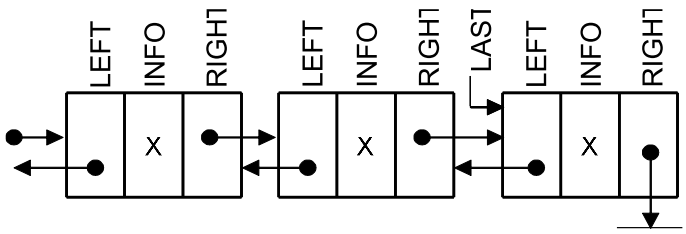
A. TUJUAN PEMBELAJARAN

Materi pada bab ini akan dijelaskan mengenai dalam memahami, menginisialisasi dan membuat algoritma Insert Pada *Linear Double Linked List*.

1. Merepresentasikan dan membuat Proses Delete data pada aplikasi *Linear doubly Linked List* dalam bahasa pemrograman

B. URAIAN MATERI

II 7.Delete Kanan



Proses dilakuakan dengan menghapus simpul yang ada dilinked list paling akhir/kanan. Fungsi:

```
void DELETE_KANAN(void)
{
        LAST=LAST->LEFT;
        Free(LAST->RIGHT);
       LAST->RIGHT=NULL;
}
```

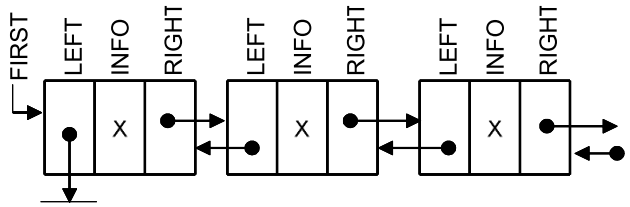
II.8. Delete Kiri

Proses dilakuakan dengan simpul yang ada pada linked list paling awal/depan.

Fungsi :

```
void DELETE_KIRI(void)
{
        FIRST=FIRST->RIGHT;
    Free(FIRST->LEFT);
    FIRST->LEFT=NULL;
}
```

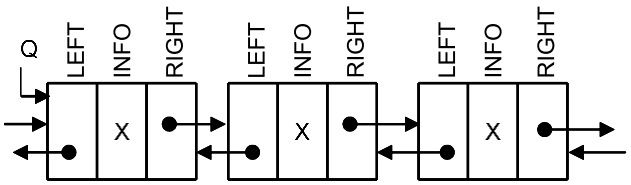
Sudah ada linked list :



II.9.Delete Tengah

Proses dilakuakan dengan menghapus simpul yang ada diantara dua simpul lain.

a. Keadaan-1 : Menghapus simpul (8) bila Q menunjuk simpul (7)



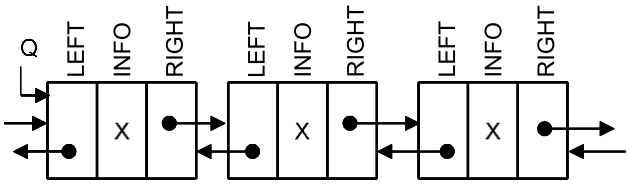
(7) (8) (9)

Fungsi :

```
void DELETE_TENGAH(void)
{
```

```
        Q->RIGHT=Q->RIGHT->RIGHT;
        free(Q->RIGHT->LEFT); Q->RIGHT-
        >LEFT=Q;
}
```

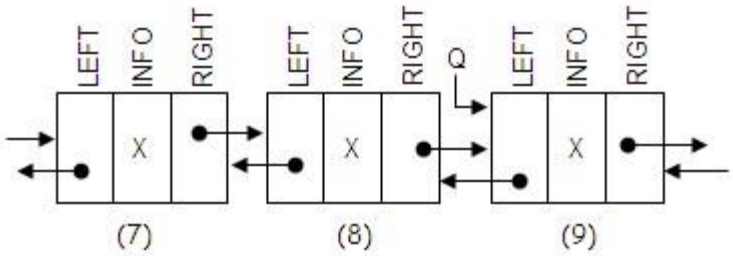
b. Keadaan-2 : Menghapus simpul (8) bila Q menunjuk simpul (8).



Fungsi :

```
void INSERT_TENGAH(void)
{
        Q->LEFT->RIGHT=Q->RIGHT; Q-
        >RIGHT->LEFT=Q->LEFT;
        free(Q);
}
```

c. Keadaan-3 : Menghapus simpul (8) bila Q menunjuk simpul (9)



Akan dihapus simpul 8 sehingga simpul 7 tersambung ke 9.

Fungsi :

```
void INSERT_TENGAH(void)
{
        Q->LEFT=Q->LEFT->LEFT;
      Free(Q->LEFT->RIGHT);
       Q->LEFT->RIGHT=Q;
}
```

C. SOAL LATIHAN/TUGAS

Latihan.12

1. Buatlah program animasi Linear Doubly Linked List untuk mengelola data mahasiswa dengan struktur komponen mahasiswa sbb : NAMA, NIM, GENDER, NILAI . Dengan data terurut naik berdasarkan NIM. Dimana program dibuat dalam bentuk menu dengan pilihan : INSERT DATA, HAPUS DATA, CETAK DATA, EXIT.

   Ket :

   INSER DATA : menambah data

   HAPUS DATA : menghapus satu data berdasarkan kriteria NIM

   CETAK DATA : mencetak seluruh isi linked list

   EXIT : Keluar/selesai

   Tampilan menu :

   LIN. DOUBLY LINKED LIST

   ==========

   1. INSERT DATA
   2. HAPUS DATA
   3. CETAK DATA 4. EXIT

   Pilihan (1 – 4) :

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Sjukani, M. (2012). *Struktur Data (Algoritma dan Struktur Data 2), Edisi 5.* Jakarta: Mitra Wacana Media.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 14
CIRCULAR SINGLY LINKED LIST

A. TUJUAN PEMBELAJARAN

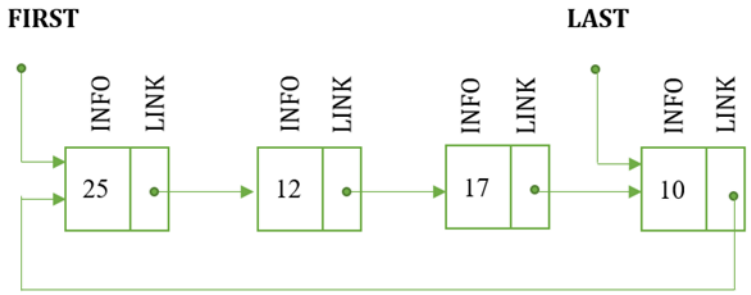
Pada bab ini akan menjelaskan mengenai proses ilustrasi dan pengaplikasian pada algoritma *Circular Singly Linked List*:

1. Mampu melakukan proses ilustrasi dan pengaplikasian pada algoritma *Circular Singly Linked List*

B. URAIAN MATERI

Mengapa Circular? Dalam singly linked list, untuk mengakses simpul mana pun dari linked list, kita mulai menelusuri dari simpul pertama. Jika kita berada di salah satu node di tengah list, maka tidak mungkin untuk mengakses node yang mendahului node yang diberikan. Masalah ini dapat diselesaikan dengan sedikit mengubah struktur singly linked list. Dalam singly linked list, bagian selanjutnya (penunjuk ke simpul berikutnya) adalah NULL, jika kita menggunakan tautan ini untuk menunjuk ke simpul pertama maka kita bisa mencapai simpul sebelumnya.

Ilustrasi

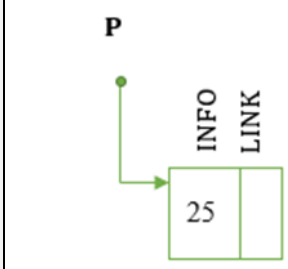


Dari ilustrasi diatas, terlihat jelas perbedaan Circular singly linked list dengan linear singly linked list yaitu pada link simpul terakhir. Maka bisa disimpulkan semua proses pada Circular singly linked list dan linear singly linked list itu sama prosesnya, kecuali pada link simpul terakhir.

Proses

1. Pembuatan untuk Sebuah Simpul

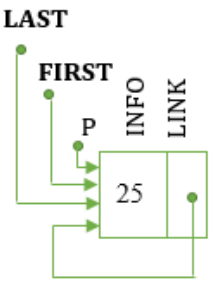
```
void BUAT_SIMPUL (int X)
{ P=(Simpul *) malloc(sizeof(Simpul));
  If ( P !=NULL)
     {
       P→INFO = X;
     }
   Else
     Cout<<"Pembuat Simpul Gagal";
```



2. Pembuatan untuk Simpul Awal

   Simpul yang sudah dibuat dijadikan sebgai Simpul Awal

```
void AWAL (int X)
{ P=(Simpul *) malloc(sizeof(Simpul));
  If ( FIRST == NULL)
     {
       FIRST = P ;
       LAST = P ;
       P →LINK = FIRST ;
     }
   Else
     Cout<<"Linked List Sudah Ada";
```



3. Insert Kanan

Fungsi untuk insert Kanan

```
Void INSERT_KANAN()
{       LAST→LINK = P ;
        FIRST = P ;
        LAST      →LINK      =
FIRST ;
}
```

4. Insert Kiri

Fungsi untuk insert Kiri

```
Void INSERT_KIRI()
{       P →LINK = FIRST ;
        FIRST = P ;
        LAST      →LINK      =
FIRST ;
}
```

5. Delete Kanan

Fungsi untuk Delete Kanan

```
Void DELETE_KANAN(void)
{       Q = FIRST;
        while (Q→LINK != LAST ;
                { Q = Q →LINK; }
        free (LAST) ;
        FIRST = Q ;
        LAST →LINK = FIRST ;
}
```

6. Delete Kiri

Fungsi untuk Delete Kiri

```
Void DELETE_KIRI(void)
{       Q = FIRST;
        FIRST = Q →LINK;
        free (Q) ;
        LAST →LINK = FIRST ;
}
```

C. SOAL LATIHAN/TUGAS

Latihan 14.

Soal 1:

Buat suatu program animasi *Circular Linked List* untuk mengelola data mahasiswa dengan struktur mahasiswa sbb : NAMA, NIM, GENDER, NILAI . Data terurut naik berdasarkan NIM. Program dibuat dalam bentuk menu dengan pilihan : INSERT DATA, HAPUS DATA, CETAK DATA, EXIT.

Ket :

INSER DATA : menambah data

HAPUS DATA : menghapus satu data berdasarkan kriteria NIM

CETAK DATA : mencetak seluruh isi linked list

EXIT : Keluar/selesai

Tampilan menu :

CIRCULAR LINKED LIST

==========

1. INSERT DATA
2. HAPUS DATA
3. CETAK DATA 4. EXIT

Pilihan (1 – 4) :

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 15
CIRCULAR DOUBLY LINKED LIST

A. TUJUAN PEMBELAJARAN

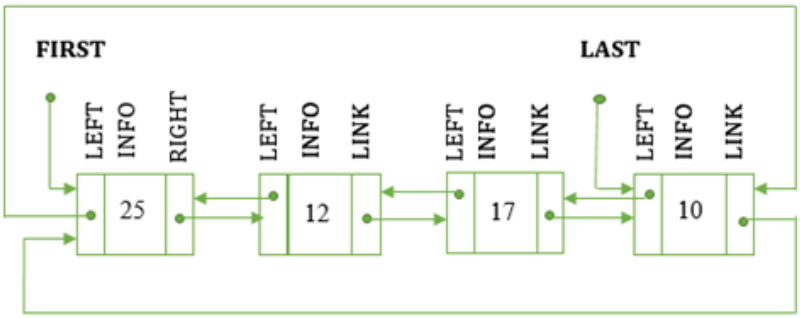
Pada bab ini akan menjelaskan mengenai proses ilustrasi dan pengaplikasian pada algoritma *Circular Doubly Linked List* :

1. Mampu melakukan proses ilustrasi dan pengaplikasian pada algoritma *Circular Doubly Linked List*

B. URAIAN MATERI

Circular Doubly Linked List memiliki properti dari *Doubly Linked List* dan Circular Doubly Linked List di mana dua elemen yang berurutan dihubungkan atau dihubungkan oleh penunjuk sebelumnya dan selanjutnya dan simpul terakhir menunjuk ke simpul pertama dengan penunjuk berikutnya dan juga titik simpul pertama menunjuk ke simpul terakhir oleh penunjuk sebelumnya.

Ilustrasi

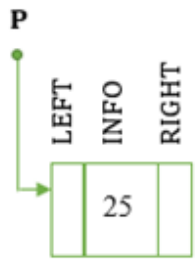


Dari ilustrasi diatas, terlihat jelas perbedaan Circular doubly linked list dengan doubly linked list dimana pointer RIGHT simpul paling kanan berisi alamat simpul paling kiri, dan pointer LEFT simpul paling kiri berisi alamat simpul paling kanan, sehingga ,menciptakan efek melingkar baik menurut arah jarum jam maupun arah sebaliknya.

Proses

1) Pembuatan untuk Sebuah Simpul

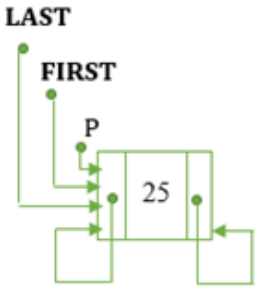
```
void BUAT_SIMPUL (int X)
{ P=(Simpul *) malloc(sizeof(Simpul));
  If ( P !=NULL)
     {
       P→INFO = X;
     }
  Else
    Cout<<"Pembuat Simpul Gagal";
```



2) Pembuatan untuk Simpul Awal

Simpul yang sudah dibuat dijadikan sebgai Simpul Awal

```
void AWAL (int X)
{ P=(Simpul *) malloc(sizeof(Simpul));
  If ( FIRST == NULL)
     {
       FIRST = P ;
       LAST = P ;
       P →RIGHT = P ;
       P →LEFT = P ;
     }
  Else
    Cout<<"Linked List Sudah Ada";
```



3) Insert Kanan

Fungsi untuk insert Kanan

```
Void INSERT_KANAN()
{       LAST→RIGHT = P ;
        LAST = P ;
        LAST →RIGHT = FIRST ;
        FIRST →LEFT = LAST ;
}
```

4) Insert Kiri

Fungsi untuk insert Kiri

```
Void INSERT_KIRI()
{       P →RIGHT = FIRST ;
        FIRST = P ;
        FIRST →LEFT = LAST ;
        LAST →RIGHT = FIRST ;
}
```

5) Delete Kanan

Fungsi untuk Delete Kanan

```
Void DELETE_KANAN(void)
{
        LAST →LAST →LEFT ;
        Free (LAST→RIGHT) ;
        FIRST →LEFT = LAST ;
        LAST →RIGHT = FIRST ;
}
```

6) Delete Kiri

Fungsi untuk Delete Kiri

```
Void DELETE_KIRI(void)
{
        FIRST = FIRST →RIGHT ;
        Free (FIRST→LEFT) ;
        FIRST →LEFT = LAST ;
        LAST →RIGHT = FIRST ;
}
```

Berikut adalah keuntungan dan kerugian dari circular doubly linked list:

Keuntungan:

- Daftar dapat dilalui dari kedua arah yaitu dari kepala ke ekor atau dari ekor ke kepala.
- Melompat dari kepala ke ekor atau dari ekor ke kepala dilakukan dalam waktu konstan O (1).
- Circular Doubly Linked List digunakan untuk implementasi struktur data tingkat lanjut seperti Fibonacci Heap .

Kekurangan

- Dibutuhkan sedikit memori ekstra di setiap node untuk mengakomodasi penunjuk sebelumnya.
- Banyak petunjuk yang terlibat saat menerapkan atau melakukan operasi pada daftar. Jadi, petunjuk harus ditangani dengan hati-hati jika tidak data dari daftar mungkin hilang.

Penerapan daftar tautan ganda melingkar

- Mengelola playlist lagu di aplikasi pemutar media.
- Mengelola keranjang belanja dalam belanja online

C. SOAL LATIHAN/TUGAS

Soal 1:

Buat program animasi Circular Doubly Linked List untuk mengelola data mahasiswa dengan struktur mahasiswa sbb : NAMA, NIM, GENDER, NILAI . Data terurut naik berdasarkan NIM. Program dibuat dalam bentuk menu dengan pilihan : INSERT DATA, HAPUS DATA, CETAK DATA, EXIT.

Ket :

INSER DATA : menambah data

HAPUS DATA : menghapus satu data berdasarkan kriteria NIM

CETAK DATA : mencetak seluruh isi linked list

EXIT : Keluar/selesai

Tampilan menu :

CIRCULAR DOUBLY LINKED LIST

==========

1. INSERT DATA
2. HAPUS DATA
3. CETAK DATA 4. EXIT

1. Pilihan (1 – 4) :

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 16
SEARCHING

A. TUJUAN PEMBELAJARAN

Bab ini akan menerangkan tentang Algoritma Searching . Melalui Penjelasan ini, Anda harus mampu:

1. Mampu memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Searching*
2. Mengenal Algoritma *Linier Search* dan *Binary Search*

B. URAIAN MATERI

Dalam ilmu komputer, pencarian adalah proses algoritmik menemukan item tertentu dalam koleksi item. Item mungkin kata kunci dalam file, catatan dalam database, node dalam pohon atau nilai dalam array dll.

1. **Mengapa Searching?**

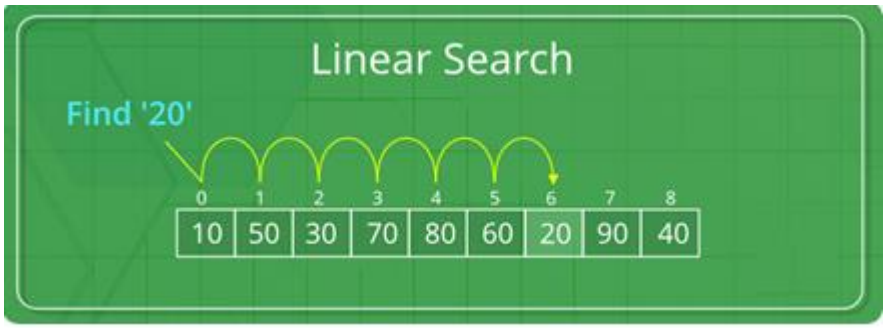
Bayangkan Anda berada di perpustakaan dengan jutaan buku. Anda ingin mendapatkan buku tertentu dengan judul tertentu. Bagaimana Anda akan menemukan? Anda hanya akan mulai mencari dengan huruf awal dari judul buku. Kemudian Anda melanjutkan pencocokan dengan judul seluruh buku sampai Anda menemukan buku yang Anda inginkan. (Dengan melakukan heuristik kecil ini Anda telah mengurangi ruang pencarian dengan faktor 26, menganggap kita memiliki jumlah buku yang sama yang judulnya dimulai dengan char tertentu.) Demikian pula, komputer menyimpan banyak informasi dan untuk mengambil informasi ini secara efisien, kita perlu algoritma pencarian yang efisien. Untuk membuat pencarian efisien, kami menyimpan data dalam beberapa urutan yang tepat. Ada cara-cara tertentu untuk mengatur data. Jika Anda menyimpan data dalam urutan yang benar, mudah untuk mencari elemen yang diperlukan. Sebagai contoh, penyortiran adalah salah satu proses untuk membuat data terorganisir.

Algoritma Pencarian dirancang untuk memeriksa elemen atau mengambil elemen dari struktur data tempat penyimpanannya. Berdasarkan jenis operasi pencarian, algoritma ini umumnya diklasifikasikan ke dalam dua kategori:

1) ***Sequential Search*** (Pencarian Berurutan): Dalam hal ini, daftar atau array dilalui secara berurutan dan setiap elemen diperiksa. Misalnya: *Linear Search*.

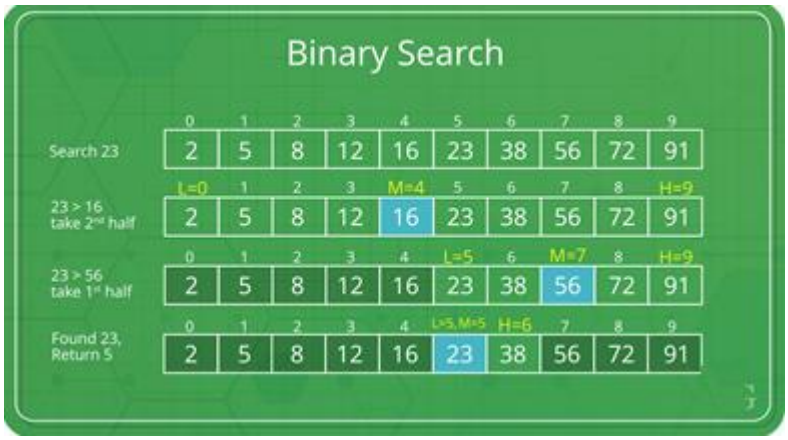
2) **Interval Search** (Pencarian Interval): Algoritme ini dirancang khusus untuk mencari dalam struktur data yang diurutkan. Jenis algoritma pencarian jauh lebih efisien daripada Linear Search karena mereka berulang kali menargetkan pusat struktur pencarian dan membagi ruang pencarian menjadi dua. Sebagai contoh: biner pencarian.

Linear Search untuk menemukan elemen "20" dalam daftar angka tertentu.



Gambar 16.1 Linear search

Binary Search untuk menemukan elemen "23" dalam daftar angka tertentu



Gambar 16.2 Binary Search

2. **Linier Search (Pencarian Linear)**

Masalah: Mengingat array arr[] dari n elemen, menulis fungsi untuk mencari elemen tertentu x in arr[]

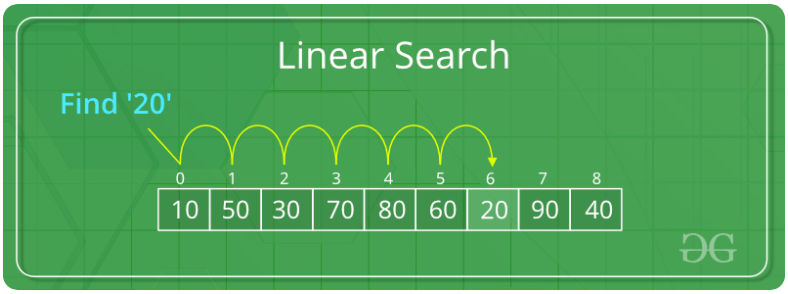
Contoh.15.1

```
Input : arr[] = {10, 20, 80, 30, 60, 50,
                 110, 100, 130, 170}
        x = 110;
Output : 6
Element x is present at index 6
```

```
Input : arr[] = {10, 20, 80, 30, 60, 50,
                 110, 100, 130, 170}
        x = 175;
Output : -1
Element x is not present in arr[].
```

Pendekatan sederhana adalah melakukan pencarian linier, yaitu

1. Mulai dari elemen paling kiri arr[] dan satu per satu bandingkan x dengan setiap elemen arr[]
2. Jika x cocok dengan elemen, kembalikan indeks.
3. Jika x tidak cocok dengan elemen apa pun, kembalikan -1.



Listing Program C++ untuk Linier Search

```
// Kode C ++ untuk mencari x secara linier di arr []. Jika x
// ada lalu kembalikan lokasinya, jika tidak
// return -1

#include <iostream>
using namespace std;

int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

// Driver code
int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    // Function call
    int result = search(arr, n, x);
    (result == -1)
        ? cout << "Elemen tidak ada dalam array"
        : cout << "Elemen ada di indeks " << result;
    return 0;
}
```

Output dari program diatas yaitu

```
Elemen ada di indeks 3
```

Listing Program C++ linear search

```
// Program C++ linear search
#include<bits/stdc++.h>
using namespace std;

void search(vector<int> arr, int search_Element)
{
    int left = 0;
    int length = arr.size();
    int position = -1;
      int right = length - 1;

    // Jalankan putaran dari 0 ke kanan
    for(left = 0; left <= right;)
    {

        // Jika search_element ditemukan dengan
        // tersisa
        if (arr[left] == search_Element)
        {

            position = left;
            cout << "Elemen ditemukan dalam Array di "
                 << position + 1 << " Posisikan dengan "
                 << left + 1 << " Attempt";

            break;
        }

        // Jika search_element ditemukan dengan
        // Variabel benar
        if (arr[right] == search_Element)
        {
            position = right;
            cout << "Elemen ditemukan dalam Array di "
```

```
                << position + 1 << " Posisikan dengan "
                << length - right << " Attempt";

            break;
        }
        left++;
        right--;
    }

    // Jika elemen tidak ditemukan
    if (position == -1)
        cout << "Tidak ditemukan di Array dengan "
             << left << " Attempt";
}

// Driver code
int main()
{
    vector<int> arr{ 1, 2, 3, 4, 5 };
    int search_element = 5;

    // Function call
    search(arr, search_element);
}
```

Output dari program diatas

```
Elemen ditemukan dalam Array di 5 Posisikan dengan 1 Attempt
```

3. **Binary Search (Pencarian Biner)**

Mengingat array arr yang diurutkan[] dari elemen n, tulis fungsi untuk mencari elemen yang diberikan x in arr[].

Pendekatan sederhana adalah melakukan pencarian linier. Kompleksitas waktu algoritma di atas adalah O(n). Pendekatan lain untuk melakukan tugas yang sama adalah menggunakan Binary Search.

Pencarian Biner: Cari array yang diurutkan dengan berulang kali membagi interval pencarian menjadi dua. Mulailah dengan interval yang mencakup seluruh array. Jika nilai kunci pencarian kurang dari item di tengah interval, sempit interval ke bagian bawah. Jika tidak mempersempit ke bagian atas. Berulang kali memeriksa sampai nilai ditemukan atau interval kosong.

Contoh:

Gambar 16.3 Binary Search

Kita pada dasarnya mengabaikan setengah dari elemen hanya setelah satu perbandingan.

1) Bandingkan x dengan elemen tengah.
2) Jika x cocok dengan elemen tengah, kita mengembalikan indeks pertengahan.
3) Else If x lebih besar dari elemen tengah, maka x hanya bisa berbaring di subarray setengah kanan setelah elemen tengah. Jadi kita berulang untuk setengah kanan.
4) Lain (x lebih kecil) kambuh untuk bagian kiri.

Implementasi rekursif Binary Search

Listing Program C++ untuk mengimplementasikan Binary Search

```
// C++ program to implement recursive Binary Search
#include <bits/stdc++.h>
using namespace std;

// A iterative binary search function. It returns
// location of x in given array arr[l..r] if present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r) {
        int m = l + (r - l) / 2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;
```

```
        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1) ? cout << "Element is not present in array"
                   : cout << "Element is present at index " << result;
    return 0;
}
```

Output dari program diatas yaitu

```
Element is present at index 3
```

Implementasi berulang dari Binary Search

```
// C++ program to implement recursive Binary Search
```

```
// C++ program to implement recursive Binary Search
#include <bits/stdc++.h>
using namespace std;


// A iterative binary search function. It returns
// location of x in given array arr[l..r] if present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r) {
        int m = l + (r - l) / 2;
```

```
        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1) ? cout << "Element is not present in array"
                   : cout << "Element is present at index " << result;
    return 0;
}
```

Output dari program diatas yaitu

```
Element is present at index 3
```

C. SOAL LATIHAN/TUGAS

Latihan 16.

1. Tunjukkan dengan tabel, Ditemukan pada iterasi ke berapa ? nilai yang diinput dari luar (missal N) ditemukan atau tidak ditemukan untuk data dan algoritma yang dicontohkan sebelumnya, bila data yang diinput yaitu:
   a. N = 17
   b. N = 22

0	1	2	3	4	5	6	7	8	9
5	7	12	15	17	19	22	25	27	32

2. Susunlah algoritma Binary Search dengan mengambil contoh data seperti pada soal no.1.

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 17
SORTING

A. TUJUAN PEMBELAJARAN

Dalam bab ini, Mampu memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Searching*. Anda harus mampu:

1. Memahami algoritma *Bubble-Sort*
2. Memahami algoritma *Insert-Sort*
3. Mengetahui algoritma *Selection-Sort*

B. URAIAN MATERI

Algoritma Penyortiran digunakan untuk mengatur ulang array atau elemen daftar tertentu sesuai dengan operator perbandingan pada elemen. Operator perbandingan digunakan untuk memutuskan urutan elemen baru dalam struktur data masing-masing.

Penyortiran adalah proses menempatkan elemen dari koleksi ke urutan naik atau turun. Sebagai contoh, ketika kita bermain kartu, menyortir kartu, sesuai dengan nilai mereka sehingga kita dapat menemukan kartu yang diperlukan dengan mudah. Ketika kita pergi ke beberapa Perpustakaan, buku diatur sesuai dengan dengan (algoritma, sistem operasi, Jaringan dll). Sorting mengatur elemen data agar pencarian menjadi lebih mudah. Ketika buku disusun dalam urutan pengindeksan yang tepat, maka mudah untuk menemukan buku yang kami Cari. Bab ini membahas algoritma untuk menyortir satu set N item. Memahami algoritma penyortiran adalah langkah pertama menuju analisis algoritma pemahaman. Banyak algoritma pengurutan dikembangkan dan dianalisis. Sebuah algoritma pengurutan seperti *Bubble-Sort, insert-Sort dan Selection-Sort* mudah diimplementasikan dan cocok untuk set input kecil. Namun, untuk dataset besar mereka lambat. Algoritma pengurutan seperti *Merge-Sort, Quick-Sort* dan *Heap-Sort* adalah beberapa dari algoritma yang cocok untuk menyortir dataset yang besar. Namun, mereka berlebihan jika kita ingin menyortir dataset kecil. Beberapa algoritma, yang cocok ketika kita memiliki beberapa informasi kisaran pada input data. Beberapa algoritma lain yang ada untuk menyortir set data yang besar yang tidak dapat disimpan dalam memori sepenuhnya, yang teknik penyortiran eksternal dikembangkan.

Contoh

Daftar karakter di bawah ini diurutkan dalam meningkatkan urutan nilai ASCII mereka. Artinya, karakter dengan nilai ASCII yang lebih rendah akan ditempatkan pertama daripada karakter dengan nilai ASCII yang lebih tinggi.

penyortiran-algoritma

1. **Type Of Sorting**

Penyortiran Internal: Semua elemen dapat dibaca ke dalam memori pada saat yang sama dan penyortiran dilakukan dalam memori.

1. Selection-Sort (Seleksi-Sortir)
2. Insertion-Sort (Penyisipan-Urutkan)
3. Bubble-Sort (Gelembung-Urutkan)
4. Quick-Sort (Pengurutan Cepat)

Penyortiran Eksternal: Dalam hal ini, dataset begitu besar sehingga tidak mungkin memuat seluruh dataset ke dalam memori sehingga penyortiran dilakukan dalam potongan.

1) Merge-Sort

Tiga hal yang perlu dipertimbangkan dalam memilih, menyortir algoritma untuk aplikasi:

1. Jumlah elemen dalam daftar
2. Sejumlah urutan yang berbeda dari daftar yang diperlukan
3. Jumlah waktu yang diperlukan untuk memindahkan data atau tidak memindahkan data

2. **Bubble-Sort**

Bubble Sort adalah algoritma penyortiran paling sederhana yang bekerja dengan berulang kali menukar elemen yang berdekatan jika mereka berada dalam urutan yang salah.

Contoh:

Pass Pertama:

(**5 1** 4 2 8) → (**1 5** 4 2 8), Di sini, algoritma membandingkan dua elemen pertama, dan swap sejak 5 > 1.

(1 **5 4** 2 8) →(1 **4 5** 2 8), Swap sejak 5 > 4

(1 4 **5 2** 8) → (1 4 **2 5** 8), Swap sejak 5 > 2

(1 4 2 **5 8**) → (1 4 2 **5 8**), Sekarang, karena elemen-elemen ini sudah dalam rangka (8 > 5), algoritma tidak menukarnya.

Pass Kedua:

(1 4 2 5 8) → (1 4 2 5 8)

(1 4 2 5 8) → (1 2 4 5 8), Swap sejak 4 > 2

(1 2 4 5 8) → (1 2 4 5 8)

(1 2 4 5 8) → (1 2 4 5 8)

Sekarang, array sudah diurutkan, tetapi algoritma kami tidak tahu apakah itu selesai. Algoritma membutuhkan satu seluruh lulus tanpa swap untuk mengetahui itu diurutkan.

Pass Ketiga:

(**1 2** 4 5 8) → (**1 2** 4 5 8)

(1 **2 4** 5 8) → (1 **2 4** 5 8)

(1 2 **4 5** 8) → (1 2 **4 5** 8)

(1 2 4 **5 8**) → (1 2 4 **5 8**)

Berikut ini adalah implementasi Bubble Sort

Listing Program C ++ untuk Bubble Short

```
// Program C ++ untuk implementasi Bubble sort
#include <bits/stdc++.h>
using namespace std;

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// Fungsi untuk mengimplementasikan bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)

    // Elemen terakhir i sudah ada
    for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}

/* Fungsi untuk mencetak array */
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Kode eksekusi
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    cout<<"Array yang diurutkan: \n";
    printArray(arr, n);
    return 0;
}
```

Output dari program diatas:

```
Array yang diurutkan:
11 12 22 25 34 64 90
```

3. **Insertion-Sort**

Kompleksitas Waktu Penyisipan-Sortir adalah O( ) yang sama dengan Bubble-Sort tetapi melakukan sedikit lebih baik dari itu. Ini adalah cara kita mengatur kartu bermain kita. Kami menjaga subarray diurutkan. Setiap nilai dimasukkan ke dalam posisi yang tepat dalam sub-array yang diurutkan di sebelah kirinya.

Insertion Sort adalah algoritma penyortiran sederhana yang bekerja mirip dengan cara Anda menyortir bermain kartu di tangan Anda. Array hampir dibagi menjadi bagian yang diurutkan dan tidak diurutkan. Nilai dari bagian yang tidak diurutkan dipilih dan ditempatkan pada posisi yang benar di bagian yang diurutkan.

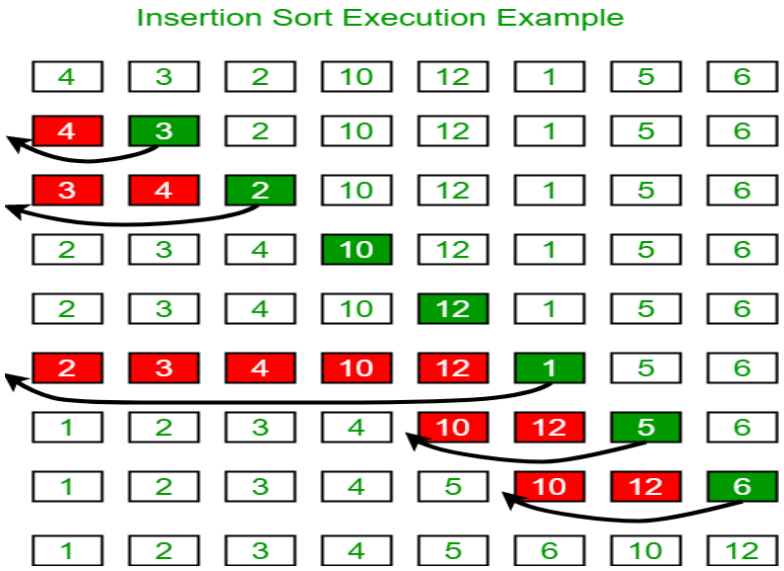
Algoritma

Untuk mengurutkan array ukuran n dalam urutan menaikan:

1. Iterate dari arr[1] untuk arr[n] atas array.

2. Bandingkan elemen saat ini (kunci) dengan pendahulunya.
3. Jika elemen kunci lebih kecil dari pendahulunya, bandingkan dengan elemen sebelumnya. Pindahkan elemen yang lebih besar satu posisi ke atas untuk membuat ruang untuk elemen swapped.

Contoh:



Contoh lain:

12, 11, 13, 5, 6

Mari kita loop untuk i = 1 (elemen kedua dari array) ke 4 (elemen terakhir dari array)

i = 1. Karena 11 lebih kecil dari 12, pindahkan 12 dan masukkan 11 sebelum 12

11, 12, 13, 5, 6

i = 2. 13 akan tetap berada di posisinya karena semua elemen dalam A[0..I-1] lebih kecil dari 13

11, 12, 13, 5, 6

i = 3. 5 akan pindah ke awal dan semua elemen lain dari 11 ke 13 akan memindahkan satu posisi di depan posisi mereka saat ini.

5, 11, 12, 13, 6

i = 4. 6 akan pindah ke posisi setelah 5, dan elemen dari 11 ke 13 akan memindahkan satu posisi di depan posisi mereka saat ini.

5, 6, 11, 12, 13

Implementasi Insertion Short

Listing Program C ++ untuk Insertion Short

```
Program C ++ untu insertion sort
#include <bits/stdc++.h>
using namespace std;

/* Fungsi untuk mengurutkan array menggunakan insertion sort*/
void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++)
   {
      key = arr[i];
      j = i - 1;

      /* Pindahkan elemen arr [0..i-1], yaitu lebih besar dari kunci, ke satu posisi di depan dari posisi saat ini */
      while (j >= 0 && arr[j] > key)
      {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}
```

```
// Fungsi utilitas untuk mencetak array berukuran n
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

/* Kode eksekusi */
int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

Output dari program diatas:

```
5 6 11 12 13
```

4. **Selection-Sort**

Selection-Sort mencari seluruh array yang tidak diurutkan dan menempatkan nilai terbesar di akhir itu. Algoritma ini memiliki kompleksitas waktu yang sama, tetapi melakukan lebih baik daripada bubble short dan penyisipan-urutkan sebagai kurang jumlah perbandingan yang diperlukan. Array diurutkan dibuat mundur dalam Selection-Sort.

Algoritma pengurutan seleksi mengurutkan array dengan berulang kali menemukan elemen minimum dari bagian yang tidak diurutkan dan

meletakkannya di awal. Algoritma mempertahankan dua subarrays dalam array tertentu.

1) Subarray yang sudah diurutkan.
2) Sisa subarray yang tidak diurutkan.

Dalam setiap iterasi jenis seleksi, elemen minimum dari subarray yang tidak diurutkan dipilih dan dipindahkan ke subarray yang diurutkan.

Implementasi Selection Sort

Listing Program C++ untuk Selection Short

```
// Program C++ untuk implementasi selection sort
#include <bits/stdc++.h>
using namespace std;

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // Satu per satu batas perpindahan subarray yang tidak disortir
    for (i = 0; i < n-1; i++)
    {
        // Temukan elemen minimum dalam array yang tidak diurutkan
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;
```

```
        // Tukar elemen minimum yang ditemukan dengan elemen pertama
        swap(&arr[min_idx], &arr[i]);
    }
}

/* Fungsi untuk mencetak sebuah array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// kode program untuk eksekusi
int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    cout << "Array yang diurutkan: \n";
    printArray(arr, n);
    return 0;
}
```

Output dari program diatas:

```
Array yang diurutkan:
11 12 22 25 64
```

C. SOAL LATIHAN/TUGAS

Latihan 17.

1. Apa kompleksitas waktu terbaik bubble short?
    a. N^2
    b. NLogN
    c. N
    d. N(LogN)^2
2. Asumsikan bahwa kita menggunakan Bubble Sort untuk mengurutkan n elemen yang berbeda dalam urutan menaik. Kapan kasus terbaik Bubble Sort terjadi?
    a. Ketika elemen diurutkan dalam urutan menaik
    b. Ketika elemen diurutkan dalam urutan menurun
    c. Ketika elemen tidak diurutkan berdasarkan urutan apa pun
    d. Tidak ada kasus terbaik untuk Bubble Sort. Selalu membutuhkan waktu O(n*n)
3. Jumlah swappings yang diperlukan untuk menyortir angka 8, 22, 7, 9, 31, 5, 13 dalam urutan menaik, menggunakan gubble short adalah
    a. 11
    b. 12
    c. 13

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

PERTEMUAN 18
SORTING (LANJUTAN)

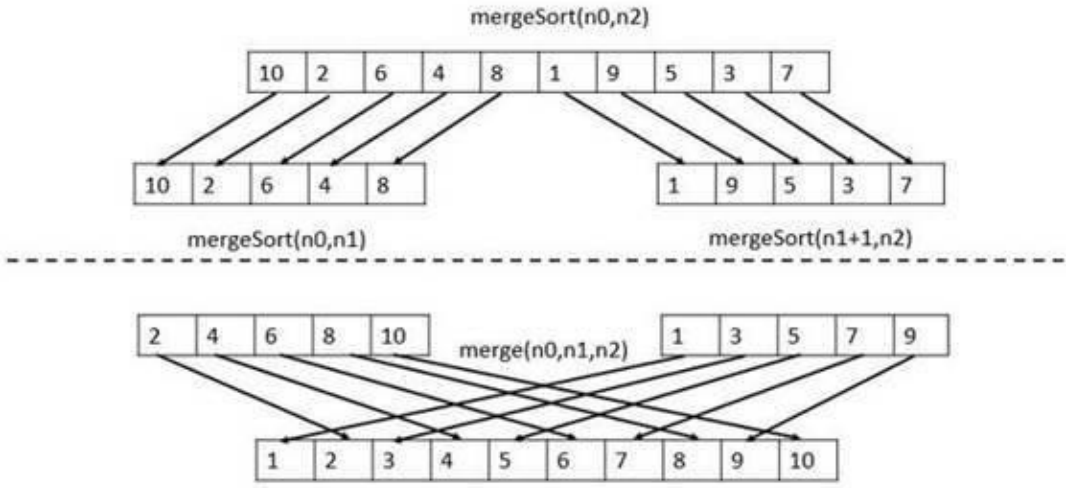
A. TUJUAN PEMBELAJARAN

Bab ini akan menjelaskan mengenai Algoritma Selection Sort dan Merge Sort dengan Baik. Melalui penjelasan ini, Anda harus mampu:

1. Mengerti algoritma Merge Sort
2. Mengerti algoritma Quick - Sort

B. URAIAN MATERI

1. Merge-Sort



Gambar 18.1 Merge Sort

Urutkan gabungan membagi input menjadi setengah rekursif di setiap langkah. Ini mengurutkan dua bagian secara terpisah secara rekursif dan akhirnya menggabungkan hasilnya menjadi output akhir yang diurutkan.

Seperti QuickSort, Merge Sort adalah algoritma Divide and Conquer. Ini membagi array input dalam dua bagian, memanggil dirinya sendiri untuk dua bagian dan kemudian menggabungkan dua bagian yang diurutkan. Fungsi merge() digunakan untuk menggabungkan dua bagian. Merge(arr, l, m, r) adalah proses kunci yang mengasumsikan bahwa arr[l.. m] dan arr[m+1..r] diurutkan dan menggabungkan dua sub-array yang diurutkan menjadi satu. Lihat implementasi C berikut untuk detailnya

MergeSort(arr[], l, r)

If r > l

1. Temukan titik tengah untuk membagi array menjadi dua bagian:

   Tengah m = (l+r)/2

2. Panggil mergeSort untuk babak pertama:

   Panggil mergeSort(arr, l, m)

3. Panggil mergeSort untuk babak kedua:

   Panggil mergeSort(arr, m+1, r)

4. Gabungkan dua bagian yang diurutkan pada langkah 2 dan 3:

   Panggil merge(arr, l, m, r)

Diagram berikut menunjukkan proses pengurutan gabungan lengkap untuk contoh array {38, 27, 43, 3, 9, 82, 10}. Jika kita melihat lebih dekat diagram, kita dapat melihat bahwa array dibagi secara rekursif dalam dua bagian sampai ukurannya menjadi 1. Setelah ukuran menjadi 1, proses penggabungan mulai beraksi dan mulai menggabungkan array kembali sampai array lengkap

Implementasi Merge Sort

Listing Program C++

```
/* Program C untuk Merge Sort */
#include <stdio.h>
#include <stdlib.h>

// Menggabungkan dua subarrays dari arr [].
// Subarray pertama adalah arr [l..m]
// Subarray kedua adalah arr [m + 1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
```

```
/* buat array temp */
int L[n1], R[n2];

/* Salin data ke array temp L [] dan R [] */
for (i = 0; i < n1; i++)
    L[i] = arr[l + i];
for (j = 0; j < n2; j++)
    R[j] = arr[m + 1 + j];

/* Salin data ke array temp L [] dan R [] */
i = 0; // Indeks awal dari subarray pertama
j = 0; // Indeks awal dari subarray kedua
k = l; // Indeks awal dari sub-array gabungan
while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    }
    else {
        arr[k] = R[j];
        j++;
    }
    k++;
}

/* Salin sisa elemen L [], jika ada */
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}
```

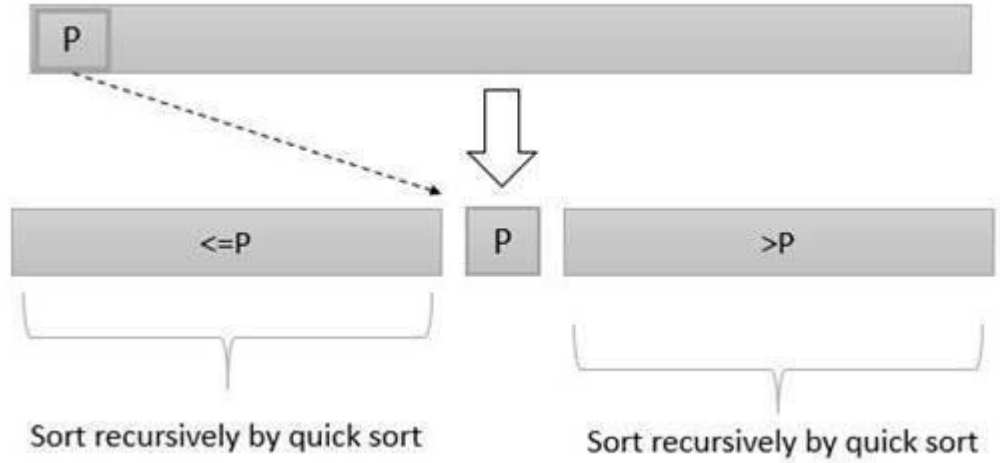
```
    /* Salin sisa elemen R [], jika ada */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l untuk indeks kiri dan r adalah indeks kanan dari sub-array arr yang akan diurutkan */
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Sama dengan (l + r) / 2, tetapi menghindari overflow for
        // besar l dan h
        int m = l + (r - l) / 2;

        // Urutkan bagian pertama dan kedua
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}
/* UTILITY FUNCTIONS */
/* Fungsi untuk mencetak array */
void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
```

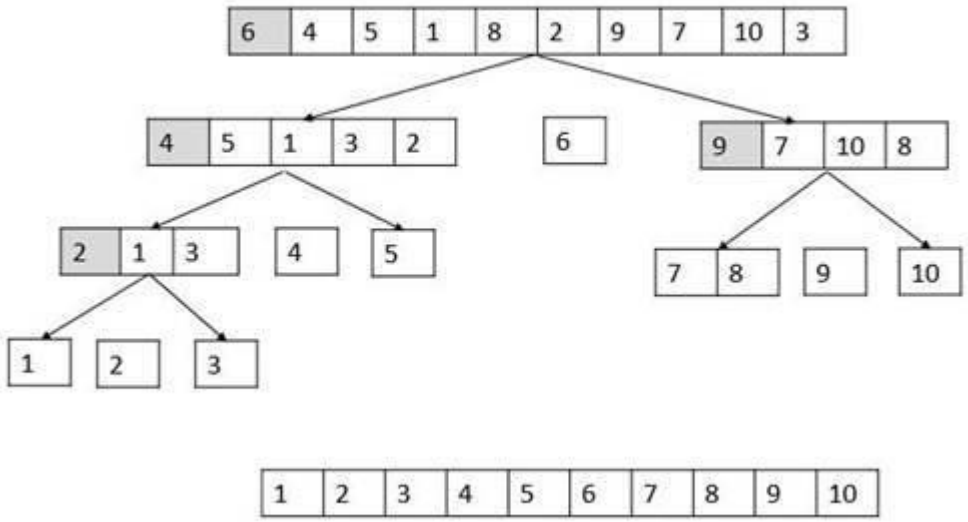
```
        printf("%d ", A[i]);
    printf("\n");
}

/* kode untuk mengeksekusi */
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Array yang diberikan adalah \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nArray yang diurutkan adalah \n");
    printArray(arr, arr_size);
    return 0;
}
```

Output dari program diatas:

```
Array yang diberikan adalah
12 11 13 5 6 7

Array yang diurutkan adalah
5 6 7 11 12 13
```

2. **Quick-Sort**





Gambar 18.2 Quick Sort

Pengurutan cepat juga merupakan algoritma rekursif. · Dalam setiap langkah kita memilih pivot (mari kita katakan elemen pertama array).

- Kemudian kita melintasi sisa array dan menyalin semua elemen array yang lebih kecil dari pivot ke sisi kiri array
- Kita menyalin semua elemen array yang grater kemudian pivot ke sisi kanan array. Jelas, pivot berada pada posisi yang diurutkan.
- Kemudian kita mengurutkan subarray kiri dan kanan secara terpisah.

- Ketika algoritma mengembalikan seluruh array diurutkan.

Seperti Merge Sort, QuickSort adalah algoritma Divide and Conquer. Ini memilih elemen sebagai pivot dan partisi array yang diberikan di sekitar pivot yang dipilih. Ada banyak versi yang berbeda dari quickSort yang memilih pivot dengan cara yang berbeda.

1) Selalu pilih elemen pertama sebagai pivot.
2) Selalu pilih elemen terakhir sebagai pivot (diterapkan di bawah)
3) Pilih elemen acak sebagai pivot.
4) Pilih median sebagai pivot.

Proses kunci dalam quickSort adalah partition(). Target partisi adalah, diberikan array dan elemen x array sebagai pivot, menempatkan x pada posisi yang benar dalam array diurutkan dan menempatkan semua elemen yang lebih kecil (lebih kecil dari x) sebelum x, dan menempatkan semua elemen yang lebih besar (lebih besar dari x) setelah x. Semua ini harus dilakukan dalam waktu linier.

Implementasi QuickShort

Listing Program C ++

```
/* Program C++ untuk QuickSort */
#include <bits/stdc++.h>
using namespace std;

// Sebuah fungsi utility untuk menukar 2 elemen
void swap(int* a, int* b)
{
   int t = *a;
   *a = *b;
   *b = t;
}

/* Fungsi ini mengambil elemen terakhir sebagai pivot, menempatkan elemen pivot pada
posisi yang benar dalam larik yang diurutkan, dan menempatkan semua yang lebih kecil
(lebih kecil dari pivot) di kiri pivot dan semua elemen yang lebih besar di sebelah kanan
pivot */
int partition (int arr[], int low, int high)
```

```
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Indeks elemen yang lebih kecil

    for (int j = low; j <= high - 1; j++)
    {

        // Jika elemen saat ini lebih kecil dari pivot
        if (arr[j] < pivot)
        {
            i++; // indeks kenaikan dari elemen yang lebih kecil
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* Fungsi utama yang mengimplementasikan QuickSort
arr [] -> Array untuk diurutkan,
rendah -> Indeks awal,
tinggi -> Indeks akhir */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi adalah indeks partisi, arr [p] sekarang di tempat yang benar */
        int pi = partition(arr, low, high);

        // Urutkan elemen sebelumnya secara terpisah
        // partisi dan setelah partisi
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
```

```
}

/* Fungsi untuk mencetak array */
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Kode untuk mengeksekusi
int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    cout << "Array yang diurutkan: \n";
    printArray(arr, n);
    return 0;
}
```

Output dari Program diatas:

```
Array yang diurutkan:
1 5 7 8 9 10
```

C. SOAL LATIHAN/TUGAS

Latihan 18.
1.

D. REFERENSI

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

GLOSARIUM

Array adalah kumpulan komponen, semua tipe yang sama, diurutkan pada N dimensi (N> = 1); setiap komponen diakses oleh N.

Biner dinyatakan dalam bentuk kombinasi angka 1 dan 0.

Binary Search adalah algoritma pencarian untuk daftar yang diurutkan yang melibatkan pembagian daftar menjadi dua dan menentukan, dengan perbandingan nilai, apakah item tersebut akan berada di atas atau setengah bagian bawah; proses ini dilakukan berulang kali hingga item ditemukan atau benar ditentukan bahwa item tersebut tidak ada dalam daftar

Body adalah pernyataan yang akan diulang dalam loop; pernyataan yang dapat dieksekusi dalam suatu fungsi

Boolean adalah tipe data yang hanya terdiri dari dua nilai: true dan false; bool di C ++

Ekspresi Boolean adalah pernyataan yang dievaluasi sebagai benar atau salah, satu-satunya nilai dari tipe data Boolean

Operator Boolean diterapkan ke nilai tipe Boolean; di C ++ ini simbol khusus &&, ||, dan!

Byte adalah delapan bit

Char adalah tipe data yang nilainya terdiri dari satu karakter alfanumerik (huruf, angka, atau simbol khusus)

Circular Linked List adalah daftar di mana setiap node memiliki penerus; elemen "terakhir" adalah digantikan oleh elemen "pertama"

Kostanta adalah item dalam program yang nilainya tetap pada waktu kompilasi dan tidak bisa berubah selama eksekusi

Constructor adalah operasi yang membuat instance baru dari tipe data abstrak (seperti file list)

Data adalah informasi yang telah dimasukkan ke dalam bentuk yang dapat digunakan komputer

Enkapsulasi Data adalah pemisahan representasi data dari aplikasi yang menggunakan data pada tingkat logis; fitur bahasa pemrograman yang memberlakukan menyembunyikan informasi

Representasi Data merupakan bentuk konkrit dari data yang digunakan untuk merepresentasikan nilai-nilai abstrak tipe data abstrak

Struktur Data adalah kumpulan elemen data yang organisasinya dicirikan oleh mengakses operasi yang digunakan untuk menyimpan dan mengambil elemen data individu; implementasi anggota data komposit dalam tipe data abstrak

Tipe Data merupakan bentuk umum kelas item data; deskripsi formal dari himpunan nilai (disebut domain) dan rangkaian operasi dasar yang dapat diterapkan padanya

Validasi Data adalah tes yang ditambahkan ke program atau fungsi yang memeriksa kesalahan dalam data

Debugging adalah proses menghilangkan kesalahan yang diketahui

Deklarasi merupakan pernyataan yang mengaitkan pengenal dengan proses atau objek sehingga pengguna dapat merujuk ke proses atau objek tersebut dengan nama

Doubly Linked List adalah linked list di mana setiap node ditautkan ke penggantinya dan pendahulunya

Ekspresi merupakan pengaturan pengidentifikasi, literal, dan operator yang dapat dievaluasi untuk menghitung nilai dari tipe tertentu

Field adalah sekelompok posisi karakter dalam satu baris keluaran

Field Identifier merupakan (pengidentifikasi anggota di C ++) nama komponen dalam record (struct)

File adalah area bernama dalam penyimpanan sekunder yang digunakan untuk menyimpan kumpulan data

Flag adalah variabel Boolean yang disetel di satu bagian program dan diuji di bagian lain untuk mengontrol aliran logis dari suatu program

Floating Point adalah nilai yang disimpan dalam variabel tipe float, disebut karena bagian dari lokasi memori menyimpan eksponen dan keseimbangan lokasi

Function adalah subprogram di C ++

Graph adalah sebuah struktur data yang terdiri dari satu set node dan satu set edge yang menghubungkan node satu sama lain

Heap adalah pohon biner lengkap, yang masing-masing elemennya berisi nilai yang lebih besar dari atau sama dengan nilai masing-masing anaknya

Identifier adalah nama yang terkait dengan proses atau objek dan digunakan untuk merujuk ke proses itu atau objek

Input adalah proses menempatkan nilai dari kumpulan data luar ke dalam variabel-variabel dalam program; data dapat berasal dari perangkat input (keyboard) atau perangkat tambahan perangkat penyimpanan (disk atau tape)

Insertion Sort adalah algoritma penyortiran di mana nilai-nilai ditempatkan satu per satu ke dalam dimana posisi yang tepat dalam daftar yang semula kosong

Linked list adalah daftar di mana urutan komponen ditentukan secara eksplisit

bidang tautan di setiap node, bukan berdasarkan urutan komponen dalam penyimpanan

Loop adalah metode penyusunan pernyataan sehingga diulangi saat kondisi tertentu terpenuhi

Parameter adalah literal, konstanta, variabel, atau ekspresi yang digunakan untuk mengkomunikasikan nilai ke atau dari subprogram

Path adalah kombinasi dari cabang-cabang yang mungkin dilintasi ketika sebuah program atau fungsi dieksekusi; urutan simpul yang menghubungkan dua simpul dalam grafik

Pointer adalah tipe data sederhana yang terdiri dari sekumpulan nilai tak terbatas, yang masing-masingnya alamat atau menunjukkan lokasi variabel dari tipe tertentu; operasi yang didefinisikan pada variabel penunjuk adalah tugas dan uji kesetaraan

Polymorphism adalah kemampuan untuk menentukan mana dari beberapa operasi yang sama nama yang sesuai; kombinasi dari ikatan statis dan dinamis

Queue adalah struktur data di mana elemen ditambahkan ke belakang dan dihapus dari depan; struktur "masuk pertama, keluar pertama" (FIFO)

Sorting adalah menyusun komponen daftar secara berurutan (misalnya, kata-kata dalam urutan abjad, angka dalam urutan menaik atau menurun)

Stack adalah tipe data abstrak di mana elemen ditambahkan dan dihapus hanya dari satu akhir; struktur "masuk terakhir, keluar pertama" (LIFO)

DAFTAR PUSTAKA

Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.

Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.

Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.

Sjukani, M. (2012). *Struktur Data (Algoritma dan Struktur Data 2), Edisi 5.* Jakarta: Mitra Wacana Media.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley.

RENCANA PEMBELAJARAN SEMESTER

(RPS)

Program Studi : S1 Teknik Informatika
Prasyarat : - Algoritma Pemrograman
Semester : 3 (Tiga)
Deskripsi Mata Kuliah : Mata Kuliah ini mengajarkan materi lanjut dari pemrograman seperti pointer, struct, dsb. Selain itu juga beberapa struktur data yang digunakan dalam pemrograman, baik yang statis atau dinamis.

Mata Kuliah/Kode : Struktur Data / TPLB21
SKS : 3 SKS
Kurikulum : KKNI
Capaian Pembelajaran : Mahasiswa mampu memecahkan masalah menjadi sebuah algoritma (langkah-langkah) yang akan dijalankan oleh komputer, kemudian mengimplementasikannya menjadi sebuah program computer sesuai kaidah ilmu pemrograman

Penyusun : 1. Achmad Udin Zailani, S.Kom., M.Kom. (Ketua)
2. Budi Apriyanto, S.Kom., M.Kom (Anggota)
3. Hadi Zakaria, S.Kom., M.Kom., M.M (Anggota)

PERTEMUAN KE-	KEMAMPUAN AKHIR YANG DIHARAPKAN	BAHAN KAJIAN (MATERI AJAR)	METODE PEMBELAJARAN	PENGALAMAN BELAJAR MAHASISWA	KRITERIA PENILAIAN	BOBOT NILAI
(1)	(2)	(3)	(4)	(5)	(6)	(7)
1	Mampu membedakan dan menerapkan beragam jenis tipe data dan hirarki data.	Tipe Data & Hirarki	Diskusi, Simulasi Dan Penugasan	Penerapan berbagai jenis tipe data dan hirarki data dalam Bahasa pemrograman	Kesesuaian penulisan dalam penggunaaan tipe data dan hirarki data dalam Bahasa pemrograman	5%

PERTEMUAN KE-	KEMAMPUAN AKHIR YANG DIHARAPKAN	BAHAN KAJIAN (MATERI AJAR)	METODE PEMBELAJARAN	PENGALAMAN BELAJAR MAHASISWA	KRITERIA PENILAIAN	BOBOT NILAI
(1)	(2)	(3)	(4)	(5)	(6)	(7)
2	Mampu menerapkan Array 1, 2 dan banyak dimensi dan tipe data pointer yang digunakan dalam Bahasa pemrograman	Array dan pointer	Diskusi, Simulasi Dan Penugasan	Menggunakan Array 1, 2 dan banyak dimensi dan data pointer dalam Bahasa pemrogramn	Kesesuaian penerapan array dan tipe data pointer dalam Bahasa pemrograman	5%
3	Mampu memahami dan menerapkan struktur, array dalam struktur dan struktur dalam struktur	Struktur	Diskusi, Simulasi Dan Penugasan	Penerapan struktur, array dalam struktur, dan struktur dalam struktur didalam Bahasa pemrograman	Kesesuaian penulisan struktur dalam Bahasa pemrograman	5%
4	Mampu memahami konsep *Single Stack,* dan algoritma PUSH dan POP pada Single *Stack*	*Single Stack*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Single Stack* dan algoritma Push dan Pop	Kesesuaian penerapan konsep *Single Stack* dan algoritma Push dan Pop dalam Bahasa pemrograman	5%
5	Mampu memahami konsep *Double Stack,* dan algoritma PUSH dan POP pada Single *Stack*	*Double Stack*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Double Stack,* dan algoritma PUSH dan POP pada *Double Stack*	Kesesuaian penerapan konsep *Double Stack* dan algoritma Push dan Pop dalam Bahasa pemrograman	5%
6	Mampu memahami, mengilustrasikan dan mengimplementasikan	*Linear Queue*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Linier Queue* dalam Bahasa Pemrograman	Kesesuaian penerapan konsep *Linier Queue* dalam	5%

PERTEMUAN KE-	KEMAMPUAN AKHIR YANG DIHARAPKAN	BAHAN KAJIAN (MATERI AJAR)	METODE PEMBELAJARAN	PENGALAMAN BELAJAR MAHASISWA	KRITERIA PENILAIAN	BOBOT NILAI
(1)	(2)	(3)	(4)	(5)	(6)	(7)
	konsep algoritma *Linear Queue*				Bahasa Pemrograman	
7	Mampu memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Circular Queue*	*Circular Queue*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Circular Queue* dalam Bahasa Pemrograman	Kesesuaian penerapan konsep *Circular Queue* dalam Bahasa Pemrograman	5%
8	Mampu memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Duble ended Queue*	*Duble ended Queue*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Duble ended Queue* dalam Bahasa Pemrograman	Kesesuaian penerapan konsep *Duble ended Queue* dalam Bahasa Pemrograman	5%
9	Mampu memahami dan merepresentasikan konsep algoritma *Liner Singly Linked List*	*Linear Singly Linked List*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Linear Singly Linked List*	Kesesuaian penerapan *Linear Singly Linked List* dalam Bahasa pemrograman	5%
UTS						
10	Mampu mengilustrasikan dan mengaplikasikan *Linked List* pada *Stack*	Aplikasi *Linked List* pada *Stack.*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Linked List* pada *Stack.*	Kesesuaian penerapan *Linked List* pada *Stack* dan Aplikasi	5%
11	Mampu mengilustrasikan dan mengaplikasikan *Single Linked List* pada *Queue*	Aplikasi *Single Linked List* pada *Queue*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Linked List* pada *Queue.*	Kesesuaian penerapan *Linked List* pada *Queue* dan Aplikasi	5%

PERTEMUAN KE-	KEMAMPUAN AKHIR YANG DIHARAPKAN	BAHAN KAJIAN (MATERI AJAR)	METODE PEMBELAJARAN	PENGALAMAN BELAJAR MAHASISWA	KRITERIA PENILAIAN	BOBOT NILAI
(1)	(2)	(3)	(4)	(5)	(6)	(7)
12	Mampu dalam memahami, menginisialisasi dan membuat algoritma Insert Pada *Linear Double Linked List*.	*Linear Double Linked List*.	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Linear Double Linked List.*	Kesesuaian penerapan *Linear Double Linked List* dan Aplikasi	5%
13	Mampu melakukan proses delete kiri, kanan dan tengah pada *Linear Double Linked List*.	*Linear Double Linked List Lanjutan*.	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Linear Double Linked List* untuk proses delete kiri, kanan dan tengah	Kesesuaian penerapan *Linear Double Linked List* untuk proses delete kiri, kanan dan tengah dan Aplikasi	5%
14	Mampu melakukan proses ilustrasi dan pengaplikasian pada algoritma *Circular Singly Linked List.*	*Circular Singly Linked List.*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Circular Singly Linked List.*	Kesesuaian penerapan *Circular Singly Linked List.* dan Aplikasi	5%
15	Mampu melakukan proses ilustrasi dan pengaplikasian pada algoritma *Circular Doubly Linked List.*	*CircularDoubly Linked List.*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Circular Doubly Linked List.*	Kesesuaian penerapan *Circular Doubly Linked List.* dan Aplikasi	5%
16	Mampu memahami, mengilustrasikan dan mengimplementasikan konsep algoritma *Searching*	*Searching*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Binary Seacrh*	Kesesuaian penerapan *Binary Seacrh* dan Aplikasi	5%

PERTEMUAN KE-	KEMAMPUAN AKHIR YANG DIHARAPKAN	BAHAN KAJIAN (MATERI AJAR)	METODE PEMBELAJARAN	PENGALAMAN BELAJAR MAHASISWA	KRITERIA PENILAIAN	BOBOT NILAI
(1)	(2)	(3)	(4)	(5)	(6)	(7)
17	Mampu memahami dan mengilustrasikan algoritma *Sorting Array*	*Sorting*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Bubble Sort, Insertion dan Selection Sort*	Kesesuaian penerapan *Bubble Sort, Insertion dan Selection Sort*	5%
18	Mampu memahami dan mengilustrasikan algoritma *Sorting Array*	*Sorting Lanjutan*	Diskusi, Simulasi Dan Penugasan	Penerapan konsep *Merge Sort dan Quick Sort*	Kesesuaian penerapan *Merge Sort dan Quick Sort* dan Aplikasi	5%
UAS						

Referensi:

1. Drozdek, A. (2012). *Data Structures and Algorithms in C++, Fourth Edition.* United States: Changeage Learning.
2. Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++, Second Edition.* United States: John Wiley & Sons, Inc.
3. Jain, H. (2016). *Problem Solving in Data Structures & Algorithms Using C++ First Edition.* CreateSpace Independent Publishing Platform.
4. Karumanchi, N. (2017). *Data Structures And Algorithms Made Easy.* Bombay: CareerMonk Publications.
5. Sjukani, M. (2012). *Struktur Data (Algoritma dan Struktur Data 2), Edisi 5.* Jakarta: Mitra Wacana Media.
6. Weiss, M. A. (2014). *Data structures and algorithm analysis in C++, Fourth Edition.* Florida: Addison Wesley

Tangerang Selatan, Sepetember 2020
Ketua Tim Teaching
Mata Kuliah Struktur Data

Achmad Udin Zailani, S.Kom., M.Kom
NIDN. 0429058303

Ketua Program Studi
S1 Teknik Informatika

Dr. Ir. Sewaka, M.M
NIDK. 8842760018