

**LAPORAN PRAKTIKUM  
STRUKTUR DATA  
SEMESTER GENAP TAHUN AKADEMIK 2021/2022**



**Disusun Oleh:**

<b>Nama</b>	<b>: Mohammad Harifin</b>
<b>NIM</b>	<b>: 2118131</b>
<b>Prodi</b>	<b>: Teknik Informatika S-1</b>
<b>Kelompok</b>	<b>: 32</b>

**PROGRAM STUDI TEKNIK INFORMATIKA S-1  
FAKULTAS TEKNOLOGI INDUSTRI  
INSTITUT TEKNOLOGI NASIONAL MALANG  
2022**

**LEMBAR PERSETUJUAN  
PRAKTIKUM STRUKTUR DATA  
SEMESTER GENAP TAHUN AKADEMIK 2021/2022**



*Disusun Oleh*

NAMA : Mohammad Harifin  
NIM : 218131  
PRODI : Teknik Informatika S-1

**Mengetahui**  
**Ka. Lab. Jaringan Komputer**      **Menyetujui**  
   **Dosen Pembimbing**

(Mira Orisa, S.T., M.T.)  
NIP. P. 1031000435

(Dedy Rudhistiar, S.Kom.M.Cs)  
NIP. P. 1032000578

**PROGRAM STUDI TEKNIK INFORMATIKA S-1  
FAKULTAS TEKNOLOGI INDUSTRI  
INSTITUT TEKNOLOGI NASIONAL MALANG  
2022**

## KATA PENGANTAR

Dengan memanjatkan puji syukur kehadirat Tuhan Yang Maha Esa, karena atas berkah rahmat dan karunia-Nya sehingga kami dapat menyelesaikan Laporan Praktikum Mata Kuliah, guna persyaratan dalam menempuh mata kuliah.

Laporan ini disusun berdasarkan percobaan dan teori dasar yang ada dalam buku panduan praktikum ,teori yang diperoleh praktikan dari perkuliahan, dan tidak lupa yaitu Internet sehingga praktikan dapat menambah tidak hanya menguasai teori saja namun juga memahami serta mengaplikasikannya.

Terwujudnya laporan ini, tentunya tidak lepas dari bantuan-bantuan yang telah kami terima. Pada kesempatan ini, kami menyampaikan terima kasih yang sebesar-besarnya kepada yang terhormat:

1. Ibu/Bapak Deddy Rudhistiar. S.Kom.M.Cs dosen pembimbing Praktikum Struktur Data.
2. Bapak Yosep Agus Pranoto,ST.,MT., Bapak Dr Agung Panji Sasmito, S.Pd., M.Pd, dan Ibu Nurlaelly Vendyansyah., S.T., M.T selaku dosen mata kuliah Struktur Data.
3. Ibu Mira Orisa,ST.MT. selaku Ketua Pelaksana Praktikum Mata Kuliah Jurusan Teknik Informatika ITN Malang.
4. Instruktur Lab. Jaringan Komputer Teknik Informatika yang telah memberi petunjuk kepada kami selama pelaksanaan praktikum.
5. Rekan-rekan yang telah membantu dalam pelaksanaan dan penyelesaian laporan ini.

Dalam menyusun laporan ini kami menyadari bahwa laporan ini masih memiliki kekurangan, karena itu segala kritik dan saran yang membangun akan kami nanti demi perbaikan penyusunan laporan selanjutnya.

Harapan kami laporan praktikum ini bermanfaat bagi penulis sendiri maupun pembaca sekalian.

Malang, Juni 2022

Penulis

## DAFTAR ISI

LEMBAR PERSETUJUAN .....	i
KATA PENGANTAR .....	ii
DAFTAR ISI .....	iii
DAFTAR GAMBAR .....	vi
DAFTAR TABEL .....	xi
BAB 1 PENDAHULUAN .....	1
1.1 Latar Belakang .....	1
1.2 Rumusan Masalah.....	1
1.3 Batasan Masalah .....	2
1.4 Tujuan .....	2
BAB 2 POINTER, STRUCTURE, REKURSIF.....	3
2.1 Jumlah Pertemuan.....	3
2.2 Tujuan .....	3
2.3 Alat dan Bahan .....	3
2.4 Landasan Teori .....	3
2.5 Tugas Praktikum ke-1: Program <i>pointer</i> .....	12
2.6 Tugas Praktikum ke-2: Program Struct .....	13
2.7 Tugas Praktikum ke-3: Program Rekursif .....	15
2.8 Kesimpulan .....	16
BAB 3 QUEUE & STACK .....	17
3.1 Jumlah Pertemuan.....	17
3.2 Tujuan .....	17
3.3 Alat dan Bahan .....	17
3.4 Landasan Teori .....	17
3.5 Tugas Praktikum ke-1: Program Menu Linear Queue.....	32

3.6 Tugas Praktikum ke-2: Perhitungan Stack.....	36
3.7 Tugas Praktikum ke-3: Program Stack .....	38
3.8 Kesimpulan .....	42
BAB 4 SORTING.....	43
4.1 Jumlah Pertemuan.....	43
4.2 Tujuan .....	43
4.3 Alat dan Bahan .....	43
4.4 Landasan Teori .....	43
4.5 Tugas Praktikum ke-1: Program Buble Sort.....	52
4.6 Tugas Praktikum ke-2: Program Exchange Sort.....	54
4.7 Tugas Praktikum ke-4: Mengurutkan Dengan Insertion Sort dan Selection.	
.....	58
4.8 Tugas Praktikum ke-3: Program Insertion Sort.....	61
4.9 Tugas Praktikum ke-5: Program Quick Sort.....	63
4.10 Kesimpulan .....	66
BAB 5 SEARCHING .....	67
5.1 Jumlah Pertemuan.....	67
5.2 Tujuan .....	67
5.3 Alat dan Bahan .....	67
5.4 Landasan Teori .....	67
5.5 Tugas Praktikum ke-1: Program Sequential Searching .....	70
5.6 Tugas Praktikum ke-2: Program Binary Searching .....	72
5.7 Tugas Praktikum ke-3: Program Interpolation Searching.....	75
5.8 Kesimpulan .....	79
BAB 6 LINKED LIST .....	80
6.1 Jumlah Pertemuan.....	80

6.2 Tujuan .....	80
6.3 Alat dan Bahan .....	80
6.4 Landasan Teori .....	80
6.5 Tugas Praktikum ke-1: Program singgel linked list.....	98
6.6 Tugas Praktikum ke-2: Doble linked list data pondok .....	104
6.7 Kesimpulan .....	111
<b>BAB 7 TREE .....</b>	<b>112</b>
7.1 Jumlah Pertemuan.....	112
7.2 Tujuan .....	112
7.3 Alat dan Bahan .....	112
7.4 Landasan Teori .....	112
7.5 Tugas Praktikum ke-1: Membuat Tree .....	118
7.6 Tugas Praktikum ke-2: Program Tree.....	123
7.7 Kesimpulan .....	130
<b>KESIMPULAN.....</b>	<b>131</b>
<b>DAFTAR PUSTAKA.....</b>	<b>132</b>

## DAFTAR GAMBAR

Gambar 2.1 Ilustrasi <i>Pointer</i> .....	4
Gambar 2.2 Penyelesaian Contoh Fungsi <i>Rekursif</i> .....	11
Gambar 2.3 Tampilan Program <i>Pointer</i> .....	12
Gambar 2.4 Tampilan Program <i>Struct</i> .....	14
Gambar 2.5 Tampilan Program permutasi.....	16
Gambar 3.1 Contoh <i>Queue</i> .....	17
Gambar 3.2 Operasi Enqueue .....	18
Gambar 3.3 Operasi Dequeue.....	18
Gambar 3.4 Operasi Is Full.....	19
Gambar 3.5 Operasi Is Empty.....	19
Gambar 3.6 Operasi Count .....	19
Gambar 3.7 Operasi Desplay .....	20
Gambar 3.8 Operasi Clear .....	20
Gambar 3.9 Implementasi <i>Queue</i> dengan <i>Circular Array</i> .....	21
Gambar 3.10 Implementasi <i>Queue</i> dengan <i>Circular Array</i> .....	21
Gambar 3.11 Pendeklarasian <i>Queue</i> .....	23
Gambar 3.12 Ilustrasi <i>Stack</i> .....	24
Gambar 3.13 Inisialisasi <i>Stack</i> .....	26
Gambar 3.14 Fungsi Is Full .....	26
Gambar 3.15 Fungsi <i>push</i> .....	26
Gambar 3.16 Fungsi <i>Pop</i> .....	27
Gambar 3.17 Fungsi <i>Print</i> .....	27
Gambar 3.18 Ilustrasi <i>Stack</i> 1 .....	28
Gambar 3.19 Ilustrasi Stack 2.....	28
Gambar 3.20 Ilustrasi <i>Stack</i> 3 .....	28

Gambar 3.21 Ilustrasi Penyelesaian <i>Stack</i> .....	29
Gambar 3.22 Tampilan Program linear queue.....	35
Gambar 3.23 Tampilan Program <i>stack</i> .....	40
Gambar 4.1 Proses 1 <i>Bubble Sort</i> .....	44
Gambar 4.2 Proses 2 <i>Bubble Sort</i> .....	44
Gambar 4.3 Proses 1 <i>Exchange Sort</i> .....	45
Gambar 4.4 Proses 2 <i>Exchange Sort</i> .....	45
Gambar 4.5 Proses 3 <i>Exchange Sort</i> .....	46
Gambar 4.6 Proses 4 <i>Exchange Sort</i> .....	46
Gambar 4.7 Proses 5 <i>Exchange Sort</i> .....	46
Gambar 4.8 Proses 1 <i>Insertion Sort</i> .....	47
Gambar 4.9 Proses 2 <i>Insertion Sort</i> .....	47
Gambar 4.10 Proses 3 <i>Insertion Sort</i> .....	47
Gambar 4.11 Proses 4 <i>Insertion Sort</i> .....	48
Gambar 4.12 Proses 5 <i>Insertion Sort</i> .....	48
Gambar 4.13 Proses 1 <i>Selection Sort</i> .....	49
Gambar 4.14 Proses 2 <i>Selection Sort</i> .....	49
Gambar 4.15 Proses 3 <i>Selection Sort</i> .....	50
Gambar 4.16 Proses 4 <i>Selection Sort</i> .....	50
Gambar 4.17 Proses 5 <i>Selection Sort</i> .....	50
Gambar 4.18 Menentukan Nilai Tengah ( <i>Quick Sort</i> ).....	51
Gambar 4.19 Tampilan Program <i>bubble sort</i> .....	53
Gambar 4.20 Tampilan Program <i>Exchange Sort</i> .....	56
Gambar 4.21 Tampilan Program <i>insertion sort</i> .....	62
Gambar 4.22 Tampilan hasil <i>Quicksort</i> .....	64
Gambar 5.1 Tampilan <i>sequential searching</i> .....	71

Gambar 5.2 Tampilan hasil <i>binary searching</i> .....	74
Gambar 5.3 Tampilan hasil <i>interpolation search</i> .....	77
Gambar 6.1 <i>Linked List</i> pada <i>computer</i> .....	80
Gambar 6.2 Ilustrasi <i>Linked List</i> .....	81
Gambar 6.3 Ilustrasi <i>Single Linked List</i> .....	81
Gambar 6.4 Ilustrasi <i>Single Linked List</i> .....	82
Gambar 6.5 <i>Linked List</i> Menggunakan <i>Head</i> .....	83
Gambar 6.6 <i>Head Null</i> .....	83
Gambar 6.7 Memasukkan 28 .....	83
Gambar 6.8 Memasukkan 21 .....	83
Gambar 6.9 <i>Head Null</i> .....	84
Gambar 6.10 Memasukkan 28 .....	84
Gambar 6.11 Memasukkan 21 .....	84
Gambar 6.12 Menampilkan isi <i>linked list</i> .....	84
Gambar 6.13 Menghapus data dari depan .....	85
Gambar 6.14 menghapus data dari belakang.....	85
Gambar 6.15 <i>Linked List</i> Menggunakan <i>Head</i> dan <i>tail</i> .....	85
Gambar 6.16 <i>Head &amp; tail Null</i> .....	86
Gambar 6.17 Memasukkan 28 .....	86
Gambar 6.18 Memasukkan 15 .....	86
Gambar 6.19 <i>Head &amp; tail Null</i> .....	86
Gambar 6.20 Memasukkan 14 .....	86
Gambar 6.21 Memasukkan 21 .....	87
Gambar 6.22 Menghapus dari depan .....	87
Gambar 6.23 Menghapus data dari belakang.....	87
Gambar 6.24 Ilustrasi <i>Double Linked List</i> .....	88

Gambar 6.25 Ilustrasi tambah <i>node</i> depan.....	93
Gambar 6.26 Ilustrasi tambah node belakang.....	94
Gambar 6.27 Ilustrasi hapus <i>node head</i> .....	96
Gambar 6.28 Ilustrasi hapus <i>node tail</i> .....	97
Gambar 6.29 Tampilan Program <i>singgel linked list</i> .....	101
Gambar 6.30 Tampilan Program doble <i>linked list</i> .....	108
Gambar 7.1 Pohon Keluarga.....	112
Gambar 7.2 Parse Tree.....	113
Gambar 7.3 Contoh Istilah Dalam Tree.....	114
Gambar 7.4 Representasi Tree.....	114
Gambar 7.5 Diagram Venn.....	114
Gambar 7.6 Notasi Tingkat.....	114
Gambar 7.7 Notasi Kurung.....	115
Gambar 7.8 Full Binary Tree .....	115
Gambar 7.9 <i>Complete Binary Tree</i> .....	115
Gambar 7.10 <i>Skewed Binary Tree</i> .....	116
Gambar 7.11 <i>Binary Search Tree</i> .....	116
Gambar 7.12 Transversing.....	117
Gambar 7.13 Pemasukan Data Pertama Ke Root .....	118
Gambar 7.14 Pembandingan Data Pertama .....	118
Gambar 7.15 Pembandingan Data .....	119
Gambar 7.16 Pemasukan Data.....	119
Gambar 7.17 Memasukan Data Sub Tree Kanan .....	119
Gambar 7.18 Memasukan Data Sub Tree Kanan .....	120
Gambar 7.19 Memasukan Data Sub Tree Kanan .....	120
Gambar 7.20 Memasukan Data Awal.....	120

Gambar 7.21 Memasukan Data Kanan .....	120
Gambar 7.22 Memasukan Sub Data Kanan.....	121
Gambar 7.23 Memasukan Data .....	121
Gambar 7.24 Memasukan Sub Data Kanan.....	121
Gambar 7.25 Memasukan Sub Data Kiri.....	122
Gambar 7.26 Memasukan Sub Data Kiri.....	122
Gambar 7.27 Hasil tampilan tree .....	129

## DAFTAR TABEL

Tabel 2.1 Operator <i>Pointer</i> .....	4
Tabel 2.2 Perbedaan <i>pointer</i> dengan variabel biasa .....	5
Tabel 2.3 Jenis Tipe Data .....	5
Tabel 3.1 Pendeklarasian <i>queue</i> .....	22
Tabel 3.2 Hasil format <i>infix</i> menjadi <i>posfik</i> .....	36
Tabel 4.1 <i>Insertion</i> proses 1.....	58
Tabel 4.2 <i>Insertion</i> proses 2.....	58
Tabel 4.3 <i>Insertion</i> proses 3.....	58
Tabel 4.4 <i>Insertion</i> proses 4.....	59
Tabel 4.5 <i>Selection</i> proses 1 .....	59
Tabel 4.6 <i>Selection</i> proses 2 .....	60
Tabel 4.7 <i>Selection</i> proses 3 .....	60
Tabel 4.8 <i>Selection</i> proses 4 .....	60
Tabel 6.1 Perbedaan <i>Linked List</i> dengan <i>Array</i> .....	81
Tabel 7.1 <i>Transversin Table ascending</i> : .....	122
Tabel 7.2 <i>Transversing Table descending</i> :.....	123

# BAB 1

## PENDAHULUAN

### 1.1 Latar Belakang

Struktur data adalah cara penyimpanan, pengorganisasian, dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien. Namun dalam bahasa pemrograman Struktur data berarti tata letak data yang berisi kolom-kolom data, baik itu kolom yang tampak oleh pengguna (user) ataupun kolom yang hanya digunakan untuk keperluan pemrograman yang tidak tampak oleh pengguna. Pemakaian struktur data yang tepat didalam proses pemrograman akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana. Konsep Struktur data ini dapat di terapkan pada bahasa pemrograman lainnya seperti pascal, dsb. Didalam struktur data ini terdapat berbagai macam pengolahan data seperti pengambilan data dengan menggunakan sebuah pointer dimana pengambilan data tersebut dengan menyebutkan alamat dari suatu variable, ada lagi yang bernama struct ialah cara pemanggilan data pada suatu fungsi yang fungsi tersebut menjadi sebuah tipedata baru di dalam sebuah program C++.

Dalam Struktur data ini program hanya bisa di pelajari secara struktural, karena program hanya akan bisa berjalan jika penulisan kode/*Script* di atas benar maka perlu penganalogian khusus seperti penganalogian pengurutan data dengan menggunakan metode *bubble sort*, *exchange sort*, dan lain sebagainya.

### 1.2 Rumusan Masalah

1. Apa itu Struktur data?
2. Bagaimana cara pemanggilan sebuah data dengan sebuah alamat?
3. Bagaimana cara pembuatan sebuah fungsi yang nantinya dapat di pakai sebagai nama variable baru?
4. Bagaimana cara pencarian sebuah data?
5. Bagaimana cara pengurutan sebuah data dari data Terbesar ke terkecil maupun sebaliknya.
6. Bagaimana cara penghitungan sebuah data dengan membedakan mana operator dan operand?

### **1.3 Batasan Masalah**

1. Mengenal pemanggilan *variable*
2. Pengurutan sebuah data
3. Pencarian sebuah data
4. Penghitungan data dengan membedakan operand dan operator

### **1.4 Tujuan**

1. Praktikan dapat mengetahui bagaimana cara pemanggilan data dengan alamat
2. Praktikan dapat mengetahui bagaimana cara pengurutan sebuah data
3. Praktikan dapat mengetahui bagaimana cara pencarian sebuah data
4. Praktikan dapat memahami bagaimana cara penghitungan sebuah data dengan bahasa tingkat tinggi

Disetujui Aslab Alfina (2018041) Tgl:	Ttd / Paraf
---	-------------

## **BAB 2**

### **POINTER, STRUCTURE, REKURSIF**

#### **2.1 Jumlah Pertemuan**

2 x 50 menit.

#### **2.2 Tujuan**

1. Praktikan mampu memahami pengertian *pointer, structure, rekursif* dengan menggunakan C++.
2. Praktikan mengetahui Aturan main dari *pointer, structure, rekursif*.
3. Praktikan dapat mengoperasikan sebuah program menggunakan metode *pointer, structure, dan rekursif*.

#### **2.3 Alat dan Bahan**

1. Perangkat komputer
2. Perangkat lunak: Dev C++
3. Modul Struktur Data 2022

#### **2.4 Landasan Teori**

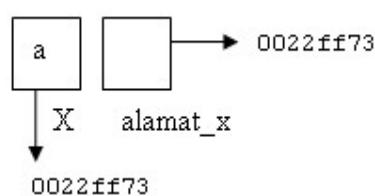
##### **A. Pointer**

###### **1) Pengertian *Pointer***

*Pointer* dapat didefinisikan sebagai suatu variabel yang menyimpan alamat memori. Jika kita mempunyai sebuah variabel dengan tipe data tertentu, maka untuk mendapatkan alamat dari variabel tersebut adalah dengan menggunakan operator & (*Dereference*). Alamat inilah yang kemudian akan disimpan ke dalam variabel yang bertipe pointer. Untuk mendeklarasikan variabel sebagai pointer, maka hanya menambahkan tanda *asterik* (\*) di depan nama variabel. Berikut ini bentuk umum dari pendeklarasian variabel yang bertipe *pointer*.

```
Tipe data *nama pointer;
```

Tipe data di atas berguna untuk menyatakan bahwa *pointer* yang kita deklarasikan tersebut akan ditempati oleh data dengan tipe tertentu.  
Ilustrasi *pointer* :



Gambar 2.1 Ilustrasi *Pointer*

Kita memiliki variabel X yang berisi nilai karakter “a”, maka oleh *compiler C++* nilai “a” ini akan disimpan di suatu alamat tertentu di memori. Sehingga alamat variabel X ini dapat diakses dengan menggunakan *statement &X*.

Jika kita ingin menyimpan alamat dari variabel X ini, kita dapat menggunakan suatu variabel misalnya

```
int alamat_x = &x
```

Maka alamat\_x adalah suatu variabel yang berisi alamat dimana nilai X disimpan. Variabel alamat\_x disebut variabel *pointer* atau sering disebut dengan *pointer* saja.

Tabel 2.1 Operator *Pointer*

Operator *	Mendapatkan nilai data dari variabel <i>pointer</i>	Contoh: <pre>int *alamat; int nilai = 10; alamat = &amp;nilai; printf("&amp;d", *alamat);</pre>	Hasil 10
Operator &	Mendapatkan alamat memori dari variabel <i>pointer</i>	Contoh: <pre>int *alamat; int nilai = 10; alamat = &amp;nilai;</pre>	Hasil 22FF70

		<code>printf ("&amp;p", *alamat);</code>	
--	--	--	--

Tabel 2.2 Perbedaan *pointer* dengan variabel biasa

Variabel Biasa	Pointer
Berisi data/nilai	Berisi alamat memori dari suatu variabel tertentu
Operasi yang bisa dilakukan seperti layaknya operasi biasa: +, -, *, /	<p>Membutuhkan operator khusus: "&amp;" yang menunjuk alamat dari suatu variabel tertentu. Operator "&amp;" hanya dapat dilakukan kepada variabel dan akan menghasilkan alamat dari variabel itu.</p> <p>Contoh: <code>p = &amp;n;</code></p> <p>Yang kedua : Operator "**". Operator ini bersifat menggunakan nilai dari alamat variabel yang ditunjuk oleh <i>pointer</i> tersebut. Contoh: <code>int *p</code></p>
Bersifat statis	Bersifat dinamis
Deklarasi: <code>int a;</code>	Deklarasi: <code>int *a;</code>

Tabel 2.3 Jenis Tipe Data

<i>float</i>	Bilangan <i>floating point</i> atau koma	4 byte	$3,4 \times 10^{-38}$ hingga $3,4 \times 10^{+38}$
<i>double</i>	Sama dengan <i>float</i> namun memiliki jangkauan dua kali dari <i>float</i>	8 byte	$1,7 \times 10^{-308}$ hingga $1,7 \times 10^{+308}$

<i>long double</i>	Sama dengan <i>double</i> namun memiliki jangkauan lebih lebar	10 byte	$3,4 \times 10^{-4932}$ hingga $3,4 \times 10^{+4932}$
<i>char</i>	Menampung tipe karakter	1 byte	-128 hingga 128
<i>int</i>	Bilangan bulat	2 byte	-32768 hingga 32768
<i>short int</i>	Sama dengan <i>int</i> namun jangkauannya lebih pendek	2 byte	-32768 hingga 32768
<i>long int</i>	Memiliki jangkauan lebih panjang dari <i>int</i>	4 byte	-2147483648 hingga 2147483648
<i>bool</i>	Tipe data untuk menampung nilai kebenaran ( <i>flag</i> )	1 byte	1 atau 0 ( <i>True</i> atau <i>False</i> )

## 2) *Pointer* Tanpa Tipe Data

Ada cara khusus untuk membuat *pointer* yang dideklarasikan tersebut dapat menunjuk ke semua tipe data, yaitu dengan mendeklarasikan *pointer* tersebut sebagai *pointer* tanpa tipe atau disebut “*void pointer*”. Bentuk umumnya adalah :

```
Void *nama_pointer
```

## 3) Aturan *Pointer*

Variabel *pointer* dapat dideklarasikan dengan tipe data apa pun. Pendeklarasian variabel *pointer* dengan tipe data tertentu digunakan untuk menyimpan alamat memori yang berisi data sesuai dengan tipe data yang dideklarasikan, bukan untuk berisi nilai bertipe data tertentu. Tipe data digunakan sebagai lebar data untuk alokasi memori (misalnya *char* berarti lebar datanya 1 byte, *Int 4 Bytes*, dst).

Jika suatu variabel *pointer* dideklarasikan bertipe *float*, berarti variabel *pointer* tersebut hanya bisa digunakan untuk menunjuk alamat memori yang berisi nilai bertipe *float* juga.

Namun besaran alokasi memori dari sebuah *pointer* dapat berbeda sesuai tipe datanya dan juga *kompiler* nya. Pada komputer yang

menggunakan 32-bit *processor*, sebuah *pointer* akan menempati 32 bits atau 4 *bytes*. Sedangkan pada komputer yang menggunakan 64-bit *processor*, *pointer* akan menempati 64 bits or 8 *bytes*. Untuk mengetahui alokasi memori pada setiap tipe data kalian dapat mencoba menjalankan perintah *size of([Tipe Data] \*)*.

#### 4) Aturan *Pointer* 2

Antar variabel *pointer* dapat dilakukan operasi *assignment*.

Contoh 1 Mengisi variabel dengan nilai yang ditunjuk oleh sebuah variabel *pointer*.

#### 5) *Pointer* Pada *Array*

*Array* adalah sebuah variabel yang menyimpan sekumpulan data yang memiliki tipe sama. setiap data tersebut menempati lokasi atau alamat *memory* yang berbeda-beda dan selanjutnya di sebut dengan *element array*. *Element array* tersebut kemudian dapat kita akses melalui indeks yang terdapat di dalamnya namun penting sekali untuk di perhatikan bahwa dalam C++, Indeks *array* selalu di mulai dari 0, bukan 1. Pada *array*, setiap *element* akan memiliki nilai dan memiliki alamat memori yang berbeda. Variabel *pointer* perlu *increment*.

## B. Struct

#### 1) Pengertian *Struct*

*Struct* (struktur) adalah kumpulan *element-element* data yang digabungkan menjadi satu kesatuan. Masing-masing elemen data tersebut dikenal dengan sebutan *field*. *Field* data tersebut dapat memiliki tipe data yang sama ataupun berbeda. Walaupun *field-field* tersebut berada dalam satu kesatuan, masing-masing *field* tersebut tetap dapat diakses secara individual.

*Field-field* tersebut digabungkan menjadi satu dengan tujuan untuk kemudahan dalam operasinya. Misalnya anda ingin mencatat data-data mahasiswa dan pelajar dalam sebuah program. Untuk membedakannya anda dapat membuat sebuah *struct* mahasiswa yang terdiri dari *field* nama, *nim*, program studi, dan *ipk*. Serta sebuah *record*

pelajar yang terdiri dari *field-field* nama, nim, alamat, dan nilai. Dengan demikian akan lebih mudah untuk membedakan keduanya.

Bentuk penulisan *struct* :

```
Struct nama_struct
{
    Tipe_data1 field1;
    Tipe_data2 field1;
    Tipe_dataN field1;
}nama_object;
```

Keterangan :

- a) *nama\_struct* : merupakan identitas dari *struct* tersebut
- b) { ..... variabel ..... } : merupakan sepasang *block*, tempat di mana semua variabel dikelompokkan sebagai member dari *struct* tersebut. Pembuatan variabel di dalam *struct* sama sekali tidak ada perbedaan dengan mendirikan variabel biasa.
- c) *nama\_object* : merupakan deklarasi yang menggunakan *struct* tersebut untuk menjadi tipe data dari deklarasi tersebut. Di tempat tersebut kita dapat membuat banyak *object*, masing-masing terpisahkan dengan tanda koma ( , ). *Object* selalu diletakan setelah penutup *block struct* dan sebelum *semicolon* ( ; ).

Contoh penulisan

```
//contoh
Struct mahasiswa //nama struct
{
    Char nim [11], nama [50];//variabel
    Char alamat [100];        //variabel
    Char ipk;                 //variabel
}Bambang; //nama_object
```

## 2) Deklarasi *Struct*

*Struct* pada dasarnya hanyalah sebuah deklarasi untuk membuat sebuah tipe data baru yang didirikan oleh *programmer* sebagai data *structure*. Data *structure* tersebut akan digunakan sebagai pembuatan *object* yang dapat dilakukan di dalam deklarasi *struct* atau *diluar* deklarasi *struct*.

Pembuatan *object* sendiri sama seperti pembuatan variabel seperti pada umumnya, yang berbeda hanya pada *struct* menggunakan tipe *structure* sebagai tipe datanya.

Pendeklarasian *object* pada *struct* :

1) Pendeklarasian *object* di dalam *struct*

Contoh deklarasi *object* di dalam deklarasi *struct*:

```
Struct mahasiswa {  
    Int nim ;  
    String nama;  
    Float ipk;  
}Bambang, suyadi; //deklarasi object di dalam deklarasi  
struct
```

2) Pendeklarasian *object* di luar *struct*

Contoh deklarasi *object* di luar *struct*:

```
Struct mahasiswa {  
    Int nim ;  
    String nama;  
    Float ipk;  
};  
Mahasiswa Bambang, suyadi; //deklarasi object di dalam  
deklarasi struct
```

3) Pengaksesan Elemen pada *Struct*

Untuk menggunakan struktur, tulis nama struktur beserta dengan *fieldnya* yang dipisahkan dengan tanda titik (“ . ”). Misalnya anda ingin menulis nim seorang mahasiswa ke layar maka penulisan yang benar adalah sebagai berikut :

- a) Mahasiswa.nim = “2018050”;
- b) cout << mahasiswa.nim;

4) *Pointer Bertype Struct*

Sebuah variabel *pointer* dapat dibuat tidak hanya untuk tipe data asli seperti (*integer*, *float*, *double*, dll) tetapi *pointer* dapat juga dibuat untuk jenis yang ditentukan pengguna seperti *structure*.

Contoh *pointer* pada *struct*:

```
Struct mahasiswa {  
    Int I;  
    Float f;  
};  
  
Int main (){  
    Mahasiswa *mhs1; //implementasi pointer pada struct
```

Jika mhs1 adalah *pointer* bertipe mahasiswa\* maka *field* dari mhs dapat diakses dengan mengganti tanda titik (“ . “) dengan tanda panah (“ -> ”). Misalnya anda ingin menulis nim seorang mahasiswa ke layar maka penulisan yang benar adalah sebagai berikut :

- a) Mahasiswa->nim = 2018050
- b) cout << mahasiswa->nim;

## C. Rekursif

### 1) Pengertian *Rekursif*

*Rekursif* adalah salah satu metode dalam dunia matematika, *rekursif* didefinisikan sebagai sebuah fungsi yang mengandung fungsi itu sendiri. Dalam dunia pemrograman, *rekursif* diimplementasikan dalam sebuah fungsi yang memanggil dirinya sendiri dan prosesnya terjadi secara berulang-ulang.

Fase dalam *rekursif* terdiri dari 3 fase yaitu :

- a) Fase Awal

Fase awal merupakan fase di mana fungsi tersebut memanggil dirinya sendiri.

- b) Fase *Terminate*

*Terminate* merupakan fase di mana fungsi tersebut berhenti memanggil dirinya sendiri.

- c) Fase Balik

Fase balik merupakan fase mengunjungi kembali kondisi – kondisi dari fase awal yang telah terbentuk dan mengembalikan nilai yang telah didapat dari fase terminal.

Contoh penerapan *rekursif*. Diketahui fungsi *rekursif* adalah :  
 $Rekursif(n) = n + Rekursif(n - 3)$

**Fase awal :**

$$Rekursif(9) = 9 + Rekursif(6)$$

$$Rekursif(6) = 6 + Rekursif(3)$$

$$Rekursif(3) = 3 + Rekursif(0)$$

$$Rekursif(0) = 0 \xrightarrow{\hspace{1cm}} \text{Terminal}$$

**Fase Balik :**

$$Rekursif(3) = 3 + 0 = 3$$

$$Rekursif(6) = 6 + 3 = 9$$

$$Rekursif(9) = 9 + 9 = 18$$

Gambar 2.2 Penyelesaian Contoh Fungsi *Rekursif*

2) Sesuatu Implementasi *Rekursif*

1) Faktorial

Dalam matematika, faktorial dari bilangan asli  $n$  adalah hasil perkalian antara bilangan bulat positif yang kurang dari atau sama dengan  $n$ . Faktorial ditulis sebagai  $n!$  dan disebut  $n$  faktorial. Secara umum dapat dituliskan sebagai:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

Di mana  $0! = 1$

Contoh : faktorial 4

$$4! = 4 \times 6 = 24$$

$$3! = 3 \times 2 = 6$$

$$2! = 2 \times 1 = 2$$

$$1! = 1$$

$$0! = 1$$

2) *Fibonacci*

*Fibonacci* adalah kumpulan deret angka seperti berikut ini “1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...”. setiap bilangan setelah bilangan kedua merupakan jumlah dari dua bilangan sebelumnya. Dengan demikian 2 dari  $1+1$ , 3 dari  $2+1$ , 5 dari  $3+2$ , dan demikian seterusnya yang merupakan definisi *rekursif* dan secara sistematis dijabarkan sebagai berikut :

Jika  $n = 0$ , maka  $F_n = 0$ , jika  $n = 1$ , maka  $F_n = 1$  Jika  $n > 1$ , maka  $F_n = F(n-1)+F(n-2)$

Implementasi dari fungsi *fibonacci* dari definisi di atas dapat dinyatakan sebagai berikut : Karena  $F_n = n$  untuk  $n < 2$ , kita dapat menyederhanakan dengan pernyataan If.

## 2.5 Tugas Praktikum ke-1: Program pointer

*Source Code :*

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

int main ()
{
    float *x;
    float a,b,c;
    a=15;
    b=25;
    c=76.4907;
    cout<<endl;
    cout<<"Nilai A : "<<a<<endl;
    cout<<"Nilai x : "<<&x<<endl;
    cout<<"Nilai &A : "<<&a<<endl;
    x=&b;
    cout<<endl;
    cout<<"Nilai B : "<<b<<endl;
    cout<<"Nilai x : "<<x<<endl;
    cout<<"Nilai &B : "<<&b<<endl;
    x=&c;
    cout<<endl;
    cout<<"Nilai C : "<<c<<endl;
    cout<<"Nilai x : "<<x<<endl;
    cout<<"Nilai &C : "<<&c<<endl;
}
```

Tampilan program :

```
File && "/Users/arifin/Documents/DATA KULIAH/
perl/"tempCodeRunnerFile

Nilai A : 15
Nilai x : 0x16cecb748
Nilai &A : 0x16cecb744

Nilai B : 25
Nilai x : 0x16cecb740
Nilai &B : 0x16cecb740

Nilai C : 76.4907
Nilai x : 0x16cecb73c
Nilai &C : 0x16cecb73c
arifin@MacBook-Air-Arifin tugas perl % █
```

Gambar 2.3 Tampilan Program Pointer

Analisa Program :

Dari program di atas yaitu membuat program *pointer* yang di mana dalam program tersebut terdapat empat variabel yang bertipe data *float* yaitu a,

b, c dan \*x. Dari tiga variabel a, b, dan c terdapat *inisialisasi* yang berbeda yaitu a=15, b=25, dan c 76.2907. Selanjutnya pada baris yang ke 12 yaitu menampilkan *output* yang udah di tentukan sebelumnya pada variabel a. Dan pada baris yang ke 13 yaitu menampilkan *output* alamat dari variabel x itu sendiri sedangkan untuk baris yang 14 yaitu menampilkan *output* dari alamat dari nilai variabel a. Untuk baris 18 yang di mana *output* nilai x akan menampilkan alamat dari nilai variabel b dengan operator yang digunakan x=&b. Dan untuk *output* pada baris 19 yaitu menampilkan alamat nilai variabel b itu sendiri. Dan untuk baris 22 sampai 24 melakukan hal yang sama seperti pada baris 17 sampai 19.

## 2.6 Tugas Praktikum ke-2: Program Struct

Source Code :

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

struct datasiswa
{
    char nim [11], nama[50];
    int tugas, uts, uas ;
    float nilai;
    // int banyak;
};

int main(){
    datasiswa data[100];
    int banyak;
    cout<<endl;
    cout<< "Mohammad Harifin | 2118131 | kelas D \n";
    cout<<"-----\n";
    cout<<"Menghitung nilai akhir mahasiswa \n";
    cout<<"-----\n";
    cout<<"Masukan Jumlah Mahasiswa : ";cin>>banyak;
    cout<<endl;
    for ( int i = 1; i <= banyak; i++)
    {
        cout<<endl;
        cout<<"NO. "<<i<<endl;
        fflush (stdin);
        cout<<"Masukkan Nama Mahasiswa\t: ";
        gets(data[i].nama);
        cout<<"Masukkan nim Mahasiswa\t: ";
        cin>>data[i].nim;
        cout<<"Masukkan nilai Tugas\t: ";
        cin>>data[i].tugas;
        cout<<"Masukkan nilai uts\t: ";
        cin>>data[i].uts;
        cout<<"Masukkan nilai UAS\t: ";
        cin>>data[i].uas;
```

```

data[i].nilai=(data[i].uts+data[i].uas+data[i].tugas)/3;
    cout<<"Nilai akhir\t\t: "<<data[i].nilai;
    cout<<endl;
}
cout<<endl;
cout<<endl;
cout<<"-----\n";
cout<<"No\tNama\t\tNim\t\tTugas\t\tuts\t\tuas\n";
cout<<"-----\n";
for (int i = 1; i <= banyak; i++)
{
    cout<<i<<"\t"<<data[i].nama<<"\t\t"<<data[i].nim<<"\t\t"<<data[i].tugas<<"\t"<<data[i].uts<<"\t"<<data[i].uas<<endl;
}

}

```

Tampilan program :

```

2/Praktikum S.Data/tugas per1/ && g++ Struct.cpp -o Struct &&
documents/DATA KULIAH/semester 2/Praktikum S.Data/tugas per1/"St
Mohammad Harifin | 2118131 |kelas D
-----
Menghitung nilai akhir mahasiswa
-----
Masukan Jumlah Mahasiswa : 2

NO. 1
warning: this program uses gets(), which is unsafe.
Masukkan Nama Mahasiswa : arifin
Masukan nim Mahasiswa : 2118131
Masukkan nilai Tugas : 90
Masukkan nilai uts : 80
Masukkan nilai UAS : 87
Nilai akhir : 85

NO. 2
Masukkan Nama Mahasiswa : aisyah
Masukan nim Mahasiswa : 2118133
Masukkan nilai Tugas : 80
Masukkan nilai uts : 92
Masukkan nilai UAS : 90
Nilai akhir : 87

-----
No      Nama        Nim       Tugas   uts   uas
1      arifin     2118131    90     80    87
2      aisyah     2118133    80     92    90
arifin@MacBook-Air-Arifin tugas per1 %

```

Gambar 2.4 Tampilan Program *Struct*

Analisa Program :

Pada program di atas terdapat program *struct* yang di mana *struct* tersebut untuk mendeklarasikan tipe data baru yaitu datamahasiswa. Di dalam *struct* tersebut dapat sekumpulan elemen-elemen yang bertipe data berbeda beda dan program tersebut mengguna *array*. Tipe data yang ada dalam struct tersebut yaitu *char* yang bervariabel nim[11], nama[50] dan integer tugas, uts,

uas dan *float* nilai. Dan pada baris 13 yaitu memanggil *struct* datamahasiswa yang diikuti *object* data[100]. Dan untuk baris yang ke 30 yang di mana penulisan programnya (cin>>data[i].nim) pada (.) itu akan memanggil variabel nim atau yang lainnya yang tercantum dalam *struct*.

## 2.7 Tugas Praktikum ke-3: Program Rekursif

*Source Code :*

```
#include <iostream>
using namespace std;
int faktorial1 (int n){
    if (n<=1){
        return 1;
    }
    else {
        return n * faktorial1 (n-1);
    }
}
int faktorial2 (int m){
    if (m<=1){
        return 1;
    }
    else {
        return m * faktorial2 (m-1);
    }
}
int main (){
    int n,r,m,permutasi;
    cout<<"\n";
    cout<<"| PROGRAM MENGHITUNG PERMUTASI |\n";
    cout<<"|                                |\n";
    cout<<"|                                |\n";
    cout<<endl;
    cout<<endl;
    cout<<"Masukkan nilai n : ";
    cin>>n;
    cout<<"Masukan nilai r : ";
    cin>>r;
    m=n-r;
    permutasi=faktorial1(n)/faktorial2(m);
    cout<<endl;
    cout<<"Hasil pemutasan 8P3 = "<<permutasi;
    cout<<endl;
    cout<<endl;
}
```

Tampilan program :

```
permutasi && "/Users/arifin/Documents/DATA KULIKUM S.Data/tugas per2/"permutasi
| PROGRAM MENGHITUNG PERMUTASI |
-----
Masukkan nilai n : 8
Masukan nilai r : 3
Hasil pemutasan 8P3 = 336
arifin@MacBook-Air-Arifin tugas per2 %
```

Gambar 2.5 Tampilan Program permutasi

Analisa Program :

Pada program di atas yang di mana program tersebut yaitu menghitung permutasi  $8P3$  yang di mana *user* akan memasuk nilai n dan nilai r. Saat *user* sudah meng inputkan pada nilai yang diminta maka yang di deklarasikan terlebih dahulu yaitu nilai n yang akan di masukan pada fungsi faktorial1 untuk mencari nilai faktorial dari nilai n. dan selanjutnya pada nilai r dengan nilai n akan dilakukan pengurangan terlebih dahulu dan setelah itu akan menjadi nilai variabel m yang akan di masukkan pada fungsi faktorial2 dan akan dilakukan pemutasan dengan cara membagi nilai dari fungsi faktorial1 dengan faktorial2 seperti pada baris yang 32.

## 2.8 Kesimpulan

1. *Pointer* dapat didefinisikan sebagai suatu variabel yang menyimpan alamat memori. Jika kita mempunyai sebuah variabel dengan tipe data tertentu, maka untuk mendapatkan alamat dari variabel tersebut adalah dengan menggunakan operator & (*Dereference*).
2. *Struct* (struktur) adalah kumpulan element-elemen data yang digabungkan menjadi satu kesatuan.
3. *Rekursif* adalah salah satu metode dalam dunia matematika, *rekursif* didefinisikan sebagai sebuah fungsi yang mengandung fungsi itu sendiri.

Disetujui Aslab Alfina (2018041) Tgl:	Ttd / Paraf
---	-------------

## BAB 3

### QUEUE & STACK

#### 3.1 Jumlah Pertemuan

2 x 50 menit.

#### 3.2 Tujuan

1. Praktikan dapat memahami pengertian dari *Queue* dan *Stack*
2. Agar praktikan mengetahui dan memahami Operasi - Operasi yang digunakan dalam metode *Queue* dan *Stack*
3. Praktikan dapat mengoperasikan atau menerapkan metode *Queue* dan *Stack* ke dalam sebuah program

#### 3.3 Alat dan Bahan

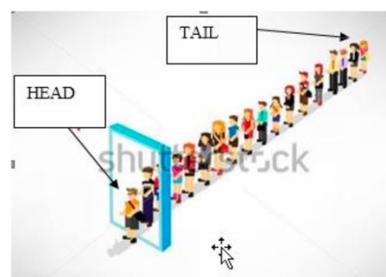
1. Perangkat komputer
2. Perangkat lunak: Dev C++
3. Modul Struktur Data 2022

#### 3.4 Landasan Teori

##### A. Queue

###### 1) Pengertian *Queue*

*Queue* atau antrian merupakan struktur data linear di mana penambahan komponen dilakukan di satu ujung, sementara pengurangan dilakukan diujung lain. Kaidah utama dalam konsep *queue* adalah FIFO yang merupakan singkatan dari First In First Out, artinya adalah data yang pertama kali dimasukkan atau disimpan, maka data tersebut adalah yang pertama kali akan diakses atau dikeluarkan.



Gambar 3.1 Contoh *Queue*

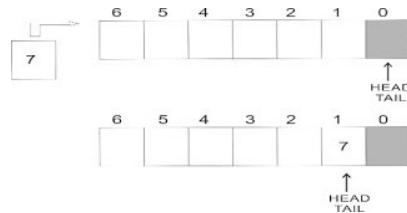
Sebuah *queue* di dalam program komputer dideklarasikan sebagai sebuah tipe bentukan baru. Sebuah struktur data dari sebuah *queue* setidaknya harus mengandung dua tiga variabel, yakni variabel

*head* yang akan berguna sebagai penanda bagian depan *antrian*, *variable tail* yang akan berguna sebagai penanda bagian belakang *antrian* dan *array* dari yang akan menyimpan data-data yang dimasukkan ke dalam *queue* tersebut.

## 2) Operasi Pada *Queue*

### a) Operasi *enqueue*

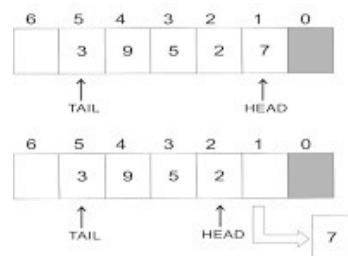
Merupakan operasi yang digunakan untuk memasukkan sebuah data atau nilai ke dalam *queue*. Pada proses *enqueue*, *tail*-lah yang berjalan seiring masuknya data baru ke dalam antrian, sedangkan *head* akan tetap pada posisi ke indeks-1.



Gambar 3.2 Operasi Enqueue

### b) Operasi *dequeue*

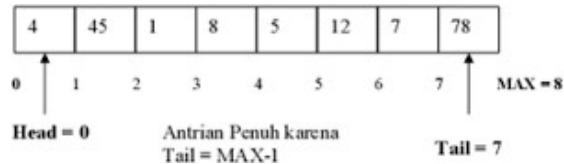
Merupakan operasi yang digunakan untuk menghapuskan sebuah data atau nilai yang paling awal masuk ke dalam *queue*. Operasi ini menaikkan menurutnkan *tail* satu level.



Gambar 3.3 Operasi Dequeue

### c) Operasi *Is Full*

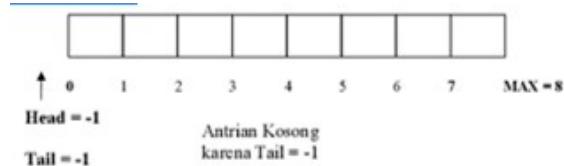
Merupakan operasi yang digunakan untuk mengecek apakah Antrian sudah penuh atau belum dengan cara mengecek nilai *Tail*, jika  $Tail \geq MAX-1$  (karena MAX-1 adalah batas elemen *array* pada C) berarti sudah penuh.



Gambar 3.4 Operasi Is Full

#### d) Operasi *IsEmpty*

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



Gambar 3.5 Operasi Is Empty

#### e) Operasi Count

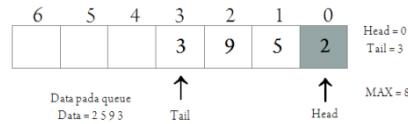
Merupakan operasi dalam *queue* yang digunakan untuk menghitung banyaknya data pada antrian. Dengan menampilkan indeks array dari *tail* yang berarti data yang ada berjumlah indeks tersebut. Contoh pada gambar 2.9 terdapat 3 kondisi, yang pertama jika kondisi *queue* kosong, yang kedua jika kondisi *queue full*, & yang ketiga jika kondisi *queue* berisikan hanya beberapa data.



Gambar 3.6 Operasi Count

#### f) Operasi Desplay

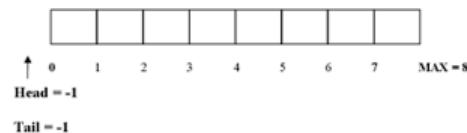
Merupakan operasi dalam *queue* yang digunakan untuk mencetak / menampilkan seluruh data yang ada pada antrian, dengan menampilkan seluruh data pada array, dari indeks *head* sampai ke indeks *tail*.



Gambar 3.7 Operasi Desplay

#### g) Operasi Clear

Untuk menghapus elemen-elemen Antrian dengan cara membuat *Tail* dan *Head* = -1, Penghapusan elemen-elemen Antrian sebenarnya tidak menghapus arraynya, namun hanya menggeser indeks pengaksesan-nya ke nilai -1 sehingga elemen- elemen Antrian tidak lagi terbaca.



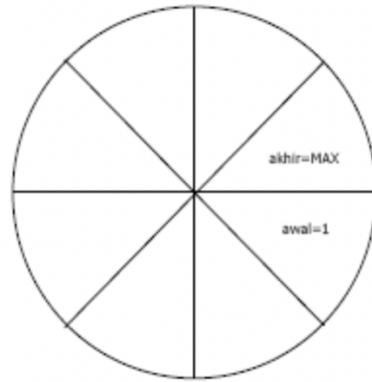
Gambar 3.8 Operasi Clear

### 3) Implementasi *Queue* Dengan Linear Array

Berikut ini diberikan deklarasi kelas *Queue* Linear sebagai implementasi dari *queue* menggunakan linear array. Dalam praktiknya, anda dapat menggantinya sesuai dengan kebutuhan anda. Data diakses dengan *field* data, sedangkan indeks item pertama dan terakhir disimpan dalam *field* *Head* dan *Tail*. *Konstruktor* akan menginisialisasikan nilai *Head* dan *Tail* dengan -1 untuk MAX\_QUEUE yang ditunjuk oleh data. Destruktor akan mengosongkan antrian kembali dan mendealokasikan memori yang digunakan oleh antrian.

### 4) Implementasi *Queue* dengan Circular Array

*Circular array* adalah suatu *array* yang dibuat seakan-akan merupakan sebuah lingkaran dengan titik awal (*head*) dan titik akhir (*tail*) saling bersebelahan jika *array* tersebut masih kosong. Posisi *head* dan *tail* pada gambar di atas adalah bebas asalkan saling bersebelahan.



Gambar 3.9 Implementasi *Queue* dengan *Circular Array*

Dengan *circular array*, meskipun posisi terakhir telah dipakai, elemen baru tetap dapat ditambahkan pada posisi pertama jika posisi pertama dalam keadaan kosong. Jika akhir = MAX dan awal = 1, nilai *head* dan *Tail* mencapai maksimum, maka akan dikembalikan ke posisi awal. Operasi- operasi yang terdapat pada *circular array* tidak jauh berbeda dengan operasi pada *linear array*.



Atau lebih jelasnya seperti gambar dibawah ini :



Gambar 3.10 Implementasi *Queue* dengan *Circular Array*

Aturan-aturan dalam *queue* yang menggunakan *circular array* adalah :

- Proses penghapusan dilakukan dengan cara nilai depan(*Head*) ditambah 1: depan += 1
- Proses penambahan elemen sama dengan *linear array* yaitu nilai belakang ditambah 1 : belakang += 1
- Jika depan = *maks* dan ada elemen yang akan dihapus, maka nilai depan = 1. Jika belakang = *maks* dan depan tidak 1, maka jika ada elemen yang akan ditambahkan, nilai belakang = 1

- d) Jika hanya tinggal 1 elemen di *queue* (depan = belakang), dan akan dihapus maka depan diisi 0 dan belakang diisi dengan 0 (*queue* kosong).

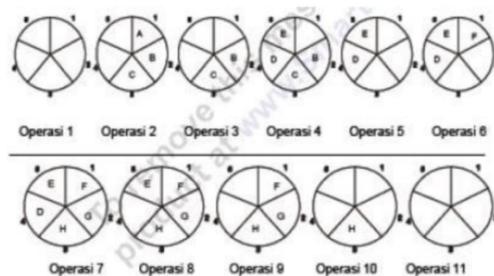
Contoh : Untuk mempermudah, elemen dari *queue* bertipe karakter.

- a) Pendeklarasian *Queue*

Tabel 3.1 Pendeklarasian *queue*

NO	Operasi	Status	Queue				
			1	2	3	4	5
1	<i>Inisialisasi</i>	Belakang = 0 Depan = 0					
2	<i>Enqueue A, B, E</i>	Depan = 1 Belakang = 3	A	B	C		
3	<i>Dequeue</i> menghasilkan nilai A	Depan = 2 Belakang = 3		B	C		
4	<i>Enqueue D, E</i>	Depan = 2 Belakang = 5		B	C	D	E
5	<i>Dequeue 2 kali</i> menghasilkan B, dan C	Depan=2 Belakang =5				D	E
6	<i>Enqueue F</i>	Depan = 4 Belakang = 1	F			D	E
7	<i>Enqueue G, H</i>	Depan = 4 Belakang = 3	F	G	H	D	E
8	<i>Dequeue</i> menghasilkan D	Depan = 5 Belakang = 4	F	G	H		E
9	<i>Dequeue</i> menghasilkan E	Depan = 1 Belakang = 3	F	G	H		

10	<i>Dequeue 2 kali menghasilkan F dan G</i>	Depan = 3 Belakang = 3			H		
11	<i>Dequeue menghasilkan H</i>	Depan = 0 Belakang = 0					



Gambar 3.11 Pendeklarasian *Queue*

Setiap *queue* memiliki elemen-elemen(*field*) berupa posisi depan, posisi belakang, elemen antrian, dan elemen maksimal.

b) Inisialisasi Queue

*Inisialisasi queue* adalah proses pemberian nilai 0 untuk *field* depan dan belakang dari *queue* dan juga pemberian nilai *maks* ke *maks\_queue* yang menunjukkan banyaknya maksimal data dalam *queue*. Karena dalam bahasa C++ elemen sebuah *array* dimulai dengan 0 maka proses *inisialisasi* nilai depan dan belakang bukan 0 tetapi -1 sehingga ketika ada proses penambahan elemen (*enqueue*) akan bernilai 0 sehingga elemen tersebut akan disimpan dalam elemen antrian pada posisi 0.

c) Fungsi Dalam Queue Circular Array

Terdapat fungsi yang mirip antara *Queue Circular* dan *Queue Linear* seperti di bawah ini :

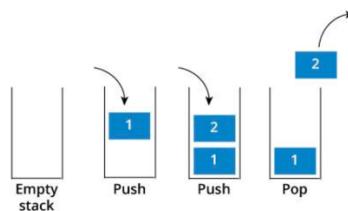
1. Fungsi Is *Empty*
2. Fungsi Is *Full*
3. Fungsi *Enqueue*
4. Fungsi *Dequeue*

## B. Stack

### 1) Pengertian *Stack*

*Stack* (Tumpukan) adalah kumpulan elemen-elemen data yang disimpan dalam satu lajur linear. Kumpulan elemen-elemen data hanya boleh diakses pada satu lokasi saja yaitu posisi ATAS (TOP) tumpukan. Tumpukan digunakan dalam algoritma pengimbas (*parsing*), algoritma penilaian (*evaluation*) dan algoritma penjajahan balik (*backtrack*). Elemen- elemen di dalam tumpukan dapat bertipe integer, real, *record* dalam bentuk sederhana atau terstruktur.

*Stack* adalah suatu tumpukan dari benda. Konsep utamanya adalah LIFO (*Last In First Out*), benda yang terakhir masuk dalam stack akan menjadi benda pertama yang dikeluarkan dari *stack*. Tumpukan disebut juga “*Push Down Stack*” yaitu penambahan elemen baru (PUSH) dan pengeluaran elemen dari tumpukan (POP). Contoh pada PDA (*Push Down Automaton*).



Gambar 3.12 Ilustrasi *Stack*

### 2) Operasi *Stack*

Operasi yang sering diterapkan pada struktur data *Stack* (Tumpukan) adalah *Push* dan *Pop*. Operasi – operasi yang dapat diterapkan adalah sebagai berikut :

- Push* : digunakan untuk menambah item pada *Stack* pada Tumpukan paling atas.
- Pop* : digunakan untuk mengambil item pada *Stack* pada Tumpukan paling atas.
- Clear* : digunakan untuk mengosongkan *Stack*.
- Create Stack* : membuat Tumpukan baru S, dengan jumlah elemen kosong.

- e) *IsEmpty* : fungsi yang digunakan untuk mengecek apakah *Stack* sudah kosong.
  - f) *Isfull* : fungsi yang digunakan untuk mengecek apakah *Stack* sudah penuh.
- 3) Macam *Stack*
- a) *Stack* dengan *Array*  
Sesuai dengan sifat *stack*, pengambilan atau penghapusan elemen dalam *stack* harus dimulai dari elemen teratas.
  - b) *Double Stack* dengan *Array*  
Metode ini adalah teknik khusus yang dikembangkan untuk menghemat pemakaian memori dalam pembuatan dua *stack* dengan *array*. Intinya adalah penggunaan hanya sebuah *array* untuk menampung dua *stack*.
- 4) Deklarasi *Stack*

Deklarasi *Stack* (Tumpukan) dengan Struktur *Array* :

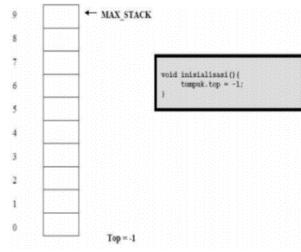
```
Struct stack
{
    Int top;
    Char data [5][5];
    //misalkan : data adalah array of string
    //berjumlah 5 data, masing-masing string
    /*jadi di sediakan data string berjumlah 5 sehingga
    dapat melakukan stack (tumpukan) sebanyak 5 tumpukan*/
};
```

Deklarasi *Stack* dengan menggunakan *Struct* :

```
#defie MAX_STACK 5
//hati-hati mulai dari 0, jadi 0-4 bukan 1-5
```

- 5) *Inisialisasi Stack*

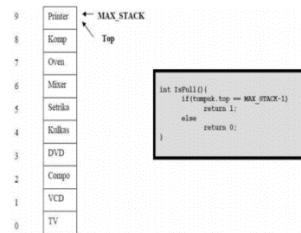
Pada mulanya isi *top* dengan -1, karena *array* dalam C dimulai dari 0, yang berarti *stack* adalah Kosong! *Top* adalah suatu variabel penanda dalam *Stack* yang menunjukkan *element* teratas *Stack* sekarang. *Top Of Stack* akan selalu bergerak hingga mencapai *Max Of Stack* sehingga menyebabkan *Stack* penuh Ilustrasi *stack* pada saat *inisialisasi* seperti pada gambar di bawah ini :



Gambar 3.13 Inisialisasi Stack

#### 6) Fungsi Is Full

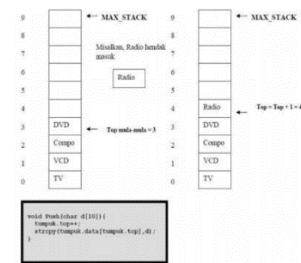
Untuk memeriksa apakah *stack* sudah penuh? Cukup dengan cara memeriksa *Top of Stack*, jika sudah sama dengan *MAX\_STACK*-1 maka dapat dinyatakan bahwa *stack* yang ada sudah penuh, tetapi jika masih lebih kecil dari *MAX\_STACK*-1 maka dapat dinyatakan bahwa *stack* belum penuh. Untuk ilustrasinya dapat dilihat seperti pada gambar di bawah ini :



Gambar 3.14 Fungsi Is Full

#### 7) Fungsi Push

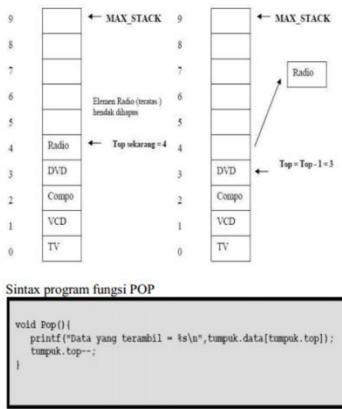
Untuk memasukkan *element* ke *stack*, selalu menjadi element teratas stack. Tambah satu (*increment*) nilai *Top of Stack* terlebih dahulu setiap kali ada penambahan *element stack*, asalkan *stack* masih belum penuh, kemudian isikan nilai baru ke *stack* berdasarkan *index Top of Stack* setelah ditambah satu.



Gambar 3.15 Fungsi push

## 8) Fungsi Pop

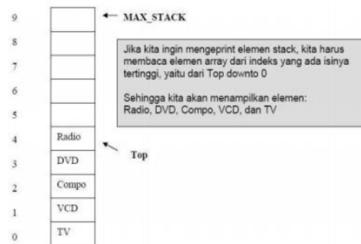
Untuk mengambil element teratas dari *stack*, ambil dahulu nilai element teratas *stack* dengan mengakses *Top of Stack*, tumpukan nilai yang akandambil terlebih dahulu, baru *didecrement* (dikurangi) nilai *Top of Stack* sehingga jumlah *element stack* berkurang seperti pada gambar di bawah ini:



Gambar 3.16 Fungsi Pop

## 9) Fungsi Print

Untuk menampilkan semua element-element *stack*. Dengan cara *looping* semua nilai *array* secara terbalik, karena kita harus mengakses dari indeks *array* tertinggi terlebih dahulu baru ke indeks yang paling kecil seperti pada gambar di bawah :

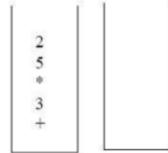


Gambar 3.17 Fungsi Print

## 10) Studi Kasus

Pembuatan kalkulator *Scientific*, misalkan operasi :  $3 + 2 * 5$ , operasi berikut disebut dengan notasi infiks. Notasi infiks tersebut harus diubah terlebih dahulu menjadi notasi *postfix*  $2\ 5\ *\ 3\ +$ . Kemudian diimplementasikan dalam *stack* sebagai berikut : *stack* soal (dalam

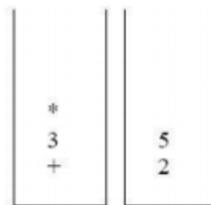
bentuk *postfix*) dan *stack* hasil (masih kosong) seperti pada gambar di bawah ini :



Gambar 3.18 Ilustrasi Stack 1

Algoritma Pop *Stack* :

- a) Jika berupa operan, maka masukkan ke *stack* hasil
- b) Jika berupa operator, maka :
  1. Pop nilai pertama dari *stack* hasil
  2. Pop nilai kedua dari *stack* hasil
  3. Lakukan operasi sesuai dengan operator yang didapat. Misalnya untuk contoh di atas pada gambar di bawah ini.



Gambar 3.19 Ilustrasi Stack 2

- c) Operator \* di pop dari *stack* soal, pop *stack* hasil dua kali, yaitu 5 dan kemudian simpan 5 ke dalam variabel misalnya a, dan 2 ke dalam variabel b. Lalu lakukan operasi sesuai dengan operatornya,  $b * a$ . Jadi  $b * a$ , yaitu  $2 * 5$ , kemudian hasilnya disimpan lagi ke dalam *stack* hasil seperti pada gambar di bawah ini :



Gambar 3.20 Ilustrasi Stack 3

Kemudian lakukan langkah yang sama sampai selesai.

Pada contoh : operator + di pop dari *stack* soal, pop *stack* hasil dua kali, yaitu 3 disimpan pada variabel a, dan 2 disimpan pada

variabel b. Kemudian dilakukan operasi sesuai dengan operatornya,  $b <operator> a$ . Jadi  $b + a$ , yaitu  $8 + 3 = 11$

Contoh, cara lain : Penghitungan :  $((1 + 2) * 4) + 3$  dapat ditulis berurut ke bawah secara *postfix* dengan keuntungan tanpa preseden pada aturan dan pokok masalahnya :  $1\ 2\ +\ 4\ *\ 3\ +$

Persamaan dimasukkan dari kiri ke kanan menggunakan *stack* :

- a) *Push* operan yang dihitung dan
- b) *Pop Two Operand* dan nilai hasil operasi penghitungan
- c) *Push* hasil penghitungan dengan langkah seperti diilustrasikan sebagai berikut ini :

Input	Operation	Stack
1	Push operand 1	
2	Push operand 1, 2	
+	Add	3
4	Push operand 3, 4	
*	Multiply	12
3	Push operand 12, 3	
+	Add	15

Gambar 3.21 Ilustrasi Penyelesaian *Stack*

Hasil akhir, 15 dan tinggalkan pada *top stack* dan selesai menghitung.

### 11) Notasi Aritmetik (*Prefix*, *Postfix*, *Infix*)

#### 1) Pengertian Notasi Aritmetik

Notasi aritmetika biasa ditulis dalam notasi *Infix*, misal  $A+B+C$ . Notasi *infix* mudah dimengerti oleh manusia, hanya saja dalam notasi *infix* perlu diperhatikan prioritas penggeraan karena berhubungan dengan hierarki operator pada komputer. Prioritas penggeraannya adalah:

1. Tanda kurung : ( ... )
  2. Eksponensial atau pangkat :  $^$
  3. Perkalian, pembagian : \*, /
  4. Penjumlahan, pengurangan : +, -. Contoh :  $(A - B) * (C + D)$
- Prioritas penggeraan soal di atas adalah sebagai berikut :
- a. Dalam kurung yang paling kiri :  $(A - B)$

- b. Dalam kurung yang kedua :  $(C - D)$
- c. Perkalian hasil pengurangan dengan hasil penjumlahan.

Notasi prefiks dan notasi *postfix* lebih mudah dikerjakan oleh komputer.

1. *Prefix* adalah keadaan di mana simbol operator diletakkan sebelum dua *operand*.
2. *Postfix* adalah keadaan di mana simbol operator diletakkan sesudah dua *operand*.

2) Aturan notasi *infix*, prefiks, dan *postfix* :

1. Notasi *Infix* : *operand operator operand*  $A + B$
2. Notasi *Prefix* : *operator operand operand*  $+ A B$  (disebut juga *Polish Notation – PN*)
3. Notasi *Postfix* : *operand operand operator* (disebut juga *Reveser Polish Notation – RPN*)

3) Contoh ke-1 : *Infix* ke *Prefix*  $(A+B)-(C*D)$

Cara penggerjaan :

1. Penggerjaan dalam kurung ke-1 :  $(A+B)$ , *prefixnya* adalah  $+AB$
2. Penggerjaan dalam kurung ke-2 :  $(C*D)$ , *prefixnya* adalah  $*CD$
3. Terakhir adalah operator  $-$ ,  $+AB - *CD$ , *prefixnya* adalah  $+AB*CD$

4) Contoh ke-2 : *Infix* ke *Postfix*  $(A+B)-(C*D)$

Cara penggerjaan :

1. Penggerjaan dalam kurung ke-1 :  $(A+B)$ , *postfixnya* adalah  $AB+$
2. Penggerjaan dalam kurung ke-2 :  $(C*D)$ , *postfixnya* adalah  $CD*$
3. Terakhir adalah operator  $-$ ,  $AB+ - CD*$ , *postfixnya* adalah  $AB+CD*-$

5) Contoh ke-3 : *Prefix* ke *Infix* :  $+/*ABCD$

Cara penggerjaan : mencari operator dimulai dari *operand* terkanan :

1. Cari operator ke-1 :  $*$ , ambil dua *operand* dari kiri A dan B, sehingga *infixnya* adalah  $(A*B)$
2. Cari operator ke-2 :  $/$ , ambil dua *operand* sebelumnya  $(A*B)$  dan C, sehingga *infixnya* adalah  $((A*B)/C)$

3. Cari operator ke-3 : +, ambil dua *operand* sebelumnya  $((A*B)/C)$  dan D, sehingga *infixnya* adalah  $((A*B)/C)+D$

6) Contoh ke-4 : *Prefix* ke *Postfix* :  $+/*ABCD$

Cara penggerjaan : mencari operator dimulai dari *operand* terkanan :

1. Cari operator ke-1 : \*, ambil dua *operand* sebelumnya A dan B, sehingga *postfixnya* adalah  $AB*$
2. Cari operator ke-2 : /, ambil dua *operand* sebelumnya  $AB*$  dan C, sehingga *postfixnya* adalah  $AB*C/$
3. Cari operator ke-3 : +, ambil dua *operand* sebelumnya  $AB*C/$  dan D, sehingga *postfixnya* adalah  $AB*C/D+$

7) Contoh ke-5 : *Postfix* ke *Infix* :  $ABCD*/-$

Cara penggerjaan : mencari operator dimulai dari *operand* terkiri :

1. Cari operator ke-1 : \*, ambil dua *operand* sebelumnya C dan D, sehingga *infixnya* adalah  $(C*D)$
2. Cari operator ke-2 : /, ambil dua *operand* sebelumnya B dan  $(C*D)$ , sehingga *infixnya* adalah  $(B/(C*D))$
3. Cari operator ke-3 : -, ambil dua *operand* sebelumnya A dan  $(B/(C*D))$ , sehingga *infixnya* adalah  $A-(B/(C*D))$

8) Contoh ke-6 : *Postfix* ke *Prefix* :  $ABCD*/-$

Cara penggerjaan : mencari operator dimulai operator terkiri :

1. Cari operator ke-1 : \*, ambil dua *operand* sebelumnya C dan D, sehingga *prefixnya* adalah  $*CD$
2. Cari operator ke-2 : /, ambil dua *operand* sebelumnya B dan  $*CD$ , sehingga *prefixnya* adalah  $/B *CD$
3. Cari operator ke-3 : -, ambil dua *operand* sebelumnya A dan  $/B *CD$ , sehingga *prefixnya* adalah  $-A/B*CD.$

## 12) Aturan Pengerjaan

- a) Baca soal dari depan ke belakang
- b) Jika soal berupa *operand*, maka masukkan ke *postfix*
- c) Jika berupa operator, maka :
  1. jika *stack* masih kosong, *push* ke *stack*
  2. jika derajat operator soal > derajat operator *top of stack*

- a) *push* operator soal ke *stack*
- 3. selama derajat operator soal  $\leq$  derajat operator top of stack
  - a) *pop top of stack* dan masukkan kedalam *postfix*
  - b) setelah semua dilakukan, *push* operator soal ke *stack*
- 4. jika sudah semua soal dibaca, pop semua isi *stack* dan *push* ke *postfix* sesuai dengan urutannya.

### 3.5 Tugas Praktikum ke-1: Program Menu Linear Queue.

*Source Code :*

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

int maxantrianarray = 5, head = -1, tail = -1;
string antrianatm[10];

bool isfull(){
    if (tail == maxantrianarray){
        return true;
    }else {
        return false;
    }
}

bool isempty(){
    if (tail == -1){
        return true;
    }else{
        return false;
    }
}

void enqueue(string data){
    if (isfull()){
        cout <<"Antrian Penuh, Tidak bisa enqueue"<<endl;
    }else{
        if(isempty()){
            antrianatm[0] = data;
            head++;
            tail++;
        }else{
            tail++;
            antrianatm[tail] = data;
        }
        cout<<data<<" telah dimasukkan antrian"<<endl;
    }
}
```

```

void dequeue() {
    if(isempty()){
        cout<<"Antrian kosong, Tidak bisa dequeue" << endl;
    }else{
        cout<<antrianatm[0];
        for(int i = 0; i < tail; i++){
            antrianatm[i] = antrianatm[i+1];
        }
        cout<<" Telah dihapus" << endl;
        tail--;
    }
}

int count(){
    if(isempty()){
        return 0;
    }else if( isfull()){// 
        return maxantrianarray;
    }else {
        return tail+1;
    }
}

void destroy(){
    if(isempty()){
        cout<<"Antrian kosong, Tidak bisa destroy" << endl;
    }else{
        for(int i = 0; i < maxantrianarray; i++){
            if(tail>1){
                antrianatm[tail-1] = "";
                tail--;
            }else if (tail == 1){
                antrianatm[tail-1] = "";
                tail=-1;
                head=-1;
            }
        }
        cout<<"Antrian di destroy" << endl;
    }
}

void display(){
    cout<<"Jumlah orang yang antri : "<<count() <<
orang." << endl;
    cout<<"Data Antrian ATM : " << endl;
    if(isempty()){
        cout<<"- [mesin ATM] -" << endl;
        cout<<"Antrian Kosong" << endl;
    }else{
        cout<<"- [mesin ATM] -" << endl;
    }
}

```

```

        for (int i = 0; i<maxantrianarray; i++) {
            if(antrianatm[i] != ""){
                cout<<i+1<<. " <<antrianatm[i]<<endl;
            }else{
                cout<<i+1<<. Kosong"<<endl;
            }
        }
        cout<<endl;
    }

int main(){
    int pilihan;
    string input;
    cout<<"2118131 | MOHAMMAD HARIFIN | D"<<endl;
    kembali:
    cout<<"======"<<endl;
    cout<<"POSISI QUEUE"<<endl;
    cout<<"Head :"<<head<<endl;
    cout<<"Tail :"<<tail<<endl;
    cout<<"======"<<endl;
    cout<<"1. masukkan Data"<<endl;
    cout<<"2. Keluarkan Data"<<endl;
    cout<<"3. Kosongkan Data"<<endl;
    cout<<"4. cetak Data Data"<<endl;
    cout<<"5. Hitung banyak data"<<endl;

    cout<<"Silahkan Masukkan Pilihan anda : ";cin>>pilihan;

    switch(pilihan){
        case 1 :
            cout<<"Masukkan Nama : ";cin>>input;
            enqueue(input);
            cout<<endl;
            goto kembali;
        break;
        case 2 :
            dequeue();
            cout<<endl;
            goto kembali;
        break;
        case 3 :
            destroy();
            cout<<endl;
            goto kembali;
        break;
        case 4 :
            display();
            cout<<endl;
            goto kembali;
        break;
    }
}

```

```

        case 5 :
            cout<<"Banyak data ada : "<<count()<<endl;
            goto kembali;
        break;
    default:
        cout<<"YANG ANDA MASUKKAN SALAH"<<endl;
    }
}

```

Tampilan program :

```

& "/Users/arifin/Documents/DATA KULIAH/semester 2/Pra
2/"Linear_Queue
2118131 | MOHAMMAD HARIFIN | D
=====
POSISI QUEUE
Head :-1
Tail :-1
=====
1. masukkan Data
2. Keluarkan Data
3. Kosongkan Data
4. cetak Data Data
5. Hitung banyak data
Silahkan Masukkan Pilihan anda : 1
Masukkan Nama : Arifin
Arifin telah dimasukkan antrian

=====
POSISI QUEUE
Head :0
Tail :0
=====
1. masukkan Data
2. Keluarkan Data
3. Kosongkan Data
4. cetak Data Data
5. Hitung banyak data
Silahkan Masukkan Pilihan anda : 1
Masukkan Nama : Ica
Ica telah dimasukkan antrian

=====
POSISI QUEUE
Head :0
Tail :1
=====
1. masukkan Data
2. Keluarkan Data
3. Kosongkan Data
4. cetak Data Data
5. Hitung banyak data
Silahkan Masukkan Pilihan anda : 2
Arifin Telah dihapus

=====
POSISI QUEUE
Head :0

```

Gambar 3.22 Tampilan Program linear queue

Analisa Program :

Pada Program di atas yang di mana program tersebut yaitu program linear *queue* dan pada program tersebut menggunakan berbagai tipe data yang salah satunya yaitu *boolean*. Pada saat program di jalankan yang dimana *user* akan diminta beberapa pilihan jika *user* memilih memasukkan data maka *user* akan meng *input* nama yang akan sebagai antrian. Dan *inputan* nama tersebut akan dimasukkan pada fungsi *void enqueue* dengan parameter (*string data*). Di dalam *enqueue* terdapat *if* dengan kondisi (*isfull()*). Di dalam *boolean isfull* terdapat sebuah *if* dengan kondisi (*tail==maxantrianarray*) untuk melakukan pengecekan apakah antrian penuh dan tidak dan jika penuh maka akan menampilkan yang sesuai pada *if* yang terpada pada fungsi *enqueue* yaitu

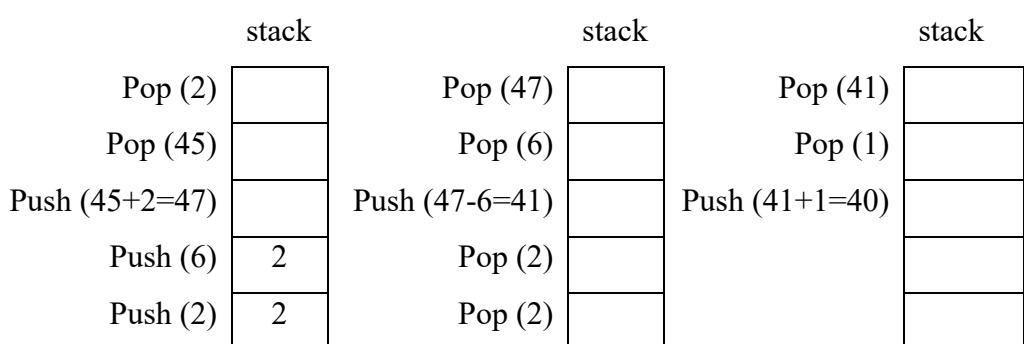
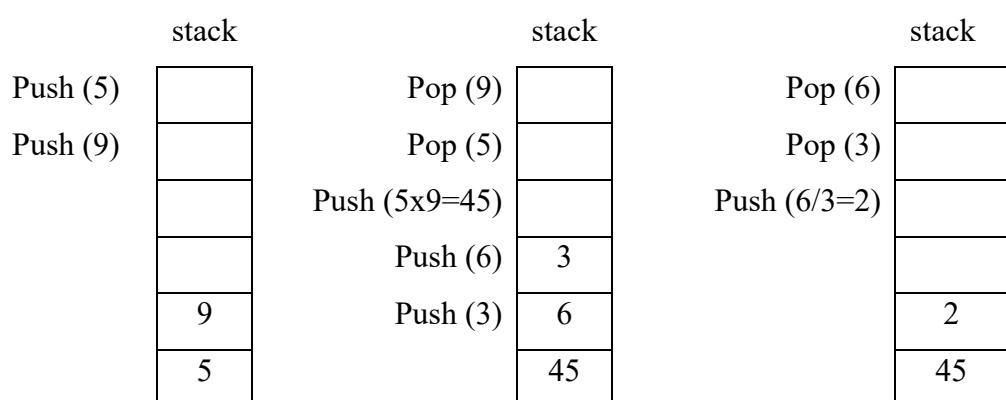
antrian penuh yang akan di tampilkan jika antrian tersebut benar-benar penuh. Jika antrian tidak penuh akan berlanjut ke else, dilam *else* terdapat if dengan kondisi (*isempty()*) dan akan langsung berlanjut ke *else* berikutnya jika datanya benar-benar kosong. Di dalam *else* terdapat perubahan *tail++* dan *antrianatm[tail]=data* dan variabel data tersebut adalah nama *inputan* dari *user* dan akan dilakukan terus menurus sampai penuh. Selanjutnya yaitu jika *user* memilih yang kedua yaitu keluarkan data maka dalam *case 2* akan melakukan pemanggilan dari fungsi *dequeue ()*. Di dalam *dequeue* terdapat if dengan kondisi (*isempty()*) yang akan melakukan pengecekan apakah *isempty true* atau *false* jika *isempty false* maka langsung berlanjut ke *else* yang dimana di *else* tersebut terdapat perulangan *for* pada baris 45 dan dalam *for* tersebut terdapat *antrianatm[i]=antrianatm[i+1]*; yang akan dilakukan pergantian seperti contoh di atas sebelumnya yang menempati *index* ke0 yaitu arifin dan *index* ke1 yaitu ica tapi jika arifin di keluarkan maka yang menempati *index* ke 0 selanjutnya yaitu ica. Selanjutnya jika *user* memilih *case ke3* yaitu kosongkan data dengan melakukan pemanggilan *destroy()*. Di dalam fungsi *destroy* melakukan pengosongan data dengan menggunakan *for* yang kan melakukan perulangan sampai data benar kosong. Jika *user* memilih cetak data yaitu *case 4* dengan melakukan pemanggilan *display()*. Di dalam *display* terdapat perulangan *for* yang akan melakukan dan menampilkan banyak data dan data yang sudah *inputkan* oleh *user* secara berulang-ulang. Selanjutnya jika *user* memilih hitung banyak data yaitu *case 5* yang akan memanggil *count()*. Di dalam *count* akan melakukan pengecekan data yang terdapat if dengan kondisi (*isempty()*) dengan di dalam nya *return 0*; yang akan otomatis jika benar data tidak ada selanjutnya *if* dengan kondisi (*isfull()*) jika antiriannya 5 maka akan di *returnkan 5* selanjutnya yaitu ke *else* yang dimana jika *user* meng *input* data sampai 3 maka yang akan di tampilkan *return 3*.

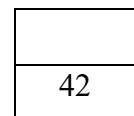
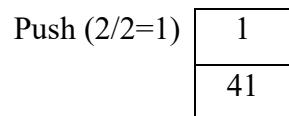
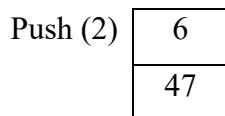
### 3.6 Tugas Praktikum ke-2: Perhitungan Stack

Tabel 3.2 Hasil format *infix* menjadi *posfix*

Masukan	Keluaran	Stack
$5 \times 9 + (6 / 3) - 6 + (2 / 2)$	Kosong	Kosong
$x 9 + (6 / 3) - 6 + (2 / 2)$	5	Kosong

$9 + (6 / 3) - 6 + (2 / 2)$	5	x
$+ (6 / 3) - 6 + (2 / 2)$	59	x
$(6 / 3) - 6 + (2 / 2)$	59x	+
$6 / 3) - 6 + (2 / 2)$	59x	+()
$/ 3) - 6 + (2 / 2)$	59x6	+()
$3) - 6 + (2 / 2)$	59x6	+(/
$) - 6 + (2 / 2)$	59x63	+(/
$- 6 + (2 / 2)$	59x63/	+
$6 + (2 / 2)$	59x63/+	-
$+ (2 / 2)$	59x63/+6	-
$(2 / 2)$	59x63/+6-	+
$2 / 2)$	59x63/+6-	+()
$/ 2)$	59x63/+6-2	+()
$2)$	59x63/+6-2	+(/
$)$	59x63/+6-22	+(/
kosong	59x63/+6-22/+	kosong





### 3.7 Tugas Praktikum ke-3: Program Stack

*Source Code :*

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

int maks = 3;
string tumpukan [3][3];
int top = 0;

// cek apakah stack penuh
bool isfull () {
    if (top == maks ) {
        return true;
    } else {
        return false;
    }
}

// cek apakah stack kosong
bool isempty () {
    if (top == 0 ) {
        return true;
    } else {
        return false;
    }
}

//push/input data dalam stack
void push (string data, string nim, string angkatan){
    if (isfull()==false){
        for (int input =0;input<=3;input++){
            if (input ==0){
                tumpukan [top][input]=data;
            }else if (input ==1){
                tumpukan [top][input]=nim;
            }if (input ==2){
                tumpukan [top][input]=angkatan;
            }
            top++;
            cout<<"Data "<<data<<" dimasukkan kedalam stack"
<<endl<<endl;
        }else {
            cout<<"Stack telah penuh !!! "<<endl<<endl;
        }
    }
}

//pop/hapus data ke dalam stack
void pop (){
    if (isempty() == false){
        for(int x=0 ; x<3;x++){
            tumpukan [top-1][x]="";
        }
    }
}
```

```

        top--;
        cout<<"Data telah di hapus" << endl;
        cout<<"-----" << endl;
    }
}

//hapus semua data

void clear(){
    if(isempty() == false){
        for(int i = maks-1; i>=0; i--){
            for(int x=0 ; x<3;x++){
                tumpukan[i][x] = "";
            }
        }
        top=0;
    }
}

//menampilkan semua data data
void print () {
    cout<<"Isi stack :" << endl << endl;
    for (int i=maks-1;i>=0;i--) {
        cout<<"-----" << endl;
        if (tumpukan [i][0]== "") {
            cout <<"Kosong" << endl;
        }
        for(int x=0 ; x<3;x++) {
            cout<<tumpukan[i][x]<< endl;
        }
    }
    cout<<"-----" << endl;
    cout<< endl;
}

int main (){
    int pil;

    string data, nim, angkatan;
    do
    {
        cout<<"Selamat datang di aplikasi stack " << endl;
        cout<<"1. PUSH (input)" << endl;
        cout<<"2. POP (hapus)" << endl;
        cout<<"3. Display (Menampilkan)" << endl;
        cout<<"4. Delete (Hapus)" << endl;
        cout<<"5. Exit " << endl;
        cout<<"Masukkan pilihan : "; cin>>pil;
        cout<<"-----"
    "<< endl;
        switch (pil) {
            case 1:

                cout<<"Masukkan nama : "; cin>>data;
                cout<<"Masukkan NIM : "; cin>>nim;
                cout<<"Masukkan Angkatan : "; cin>>angkatan;

```

```

        //cout<<"-----"
-><<endl;
        push (data,nim,angkatan);
    break;
    case 2:
        pop ();
    break;
    case 3 :
        print ();
    break;
    case 4 :
        clear();
    break;

}

}

while (pil!=5);
cout<<"Terima Kasih "<<endl;
cout<<"-----"
-><<endl;
    return 0;
}

```

Tampilan program :

```

semester 2/Praktikum S.Data/per3/" && g++ tempCodeRunnerFile
mpCodeRunnerFile && "/Users/arifin/Documents/DATA KULIAH/sem
aktikum S.Data/per3/"tempCodeRunnerFile
Selamat data di aplikasi stack
1. PUSH (input)
2. POP (hapus)
3. Display (Menampilkan)
4. Delete (Hapus)
5. Exit
Masukan pilihan : 1
-----
Masukkan nama : Arifin
Masukkan NIM : 2118131
Masukkan Angkatan : 21
Data Arifin dimasukkan kedalam stack

Selamat data di aplikasi stack
1. PUSH (input)
2. POP (hapus)
3. Display (Menampilkan)
4. Delete (Hapus)
5. Exit
Masukan pilihan : 1
-----
Masukkan nama : Aisyah
Masukkan NIM : 1811831
Masukkan Angkatan : 18
Data Aisyah dimasukkan kedalam stack

Selamat data di aplikasi stack
1. PUSH (input)
2. POP (hapus)
3. Display (Menampilkan)
4. Delete (Hapus)
5. Exit
Masukan pilihan : 3
-----
Isi stack :
-----
Kosong

```

Gambar 3.23 Tampilan Program *stack*

### Analisa Program :

Analisa pada program di atas yaitu menggunakan *array* 2 dimensi yang di mana program tersebut saat di jalankan akan menampilkan beberapa pilihan yang di pilih oleh *user*. Di dalam pilihan tersebut yaitu menggunakan *do while* untuk menjalankan pilihan *push (input)*, *pop (hapus)*, *display (menampilkan)*, *delete (hapus)* dan *exit* yang telah di tetapkan. Jika *user* memilih pilihan satu maka akan mengarah ke case 1 yang di mana *user* akan meng *input* nama, nim, angkatan dan memanggil *push()* yang di mana akan melakukan pendeklarasian *inputan user* yang sebelumnya untuk dimasukkan ke array 2 dimensi. Di dalam *if* terdapat sebuah kondisi yang akan mengecek apakah data penuh atau tidak, jika tidak maka *if* akan menjalankan perulangan *for* yang di mana dalam *for* tersebut yang akan melakukan pendeklarasian *inputan user* yang sebelumnya. Jika *user* memilih pilihan dua maka akan menuju ke *case 2* dan akan melakukan pemanggilan *pop ()* untuk melakukan penghapusan data yang sebelumnya di *input*. Jika *user* memilih 3 maka akan menuju ke *case 3* yang di mana *case 3* tersebut melakukan pemanggilan *print()* yang telah ditetapkan untuk menampilkan semua data. Jika *user* memilih ke 4 maka akan menuju ke *case 4* yang di mana akan melakukan penghapusan semua data dengan memanggil *clear()* yang telah ditentukan untuk menghapus semua data yang telah diinputkan oleh *user*. Sedangkan jika *user* memilih pilihan 5 maka menandakan program berakhir.

### 3.8 Kesimpulan

1. *Queue* atau antrian merupakan struktur data linear di mana penambahan komponen dilakukan di satu ujung, sementara pengurangan dilakukan diujung lain.
2. *Stack* (Tumpukan) adalah kumpulan elemen-elemen data yang disimpan dalam satu lajur linear. Kumpulan elemen-elemen data hanya boleh diakses pada satu lokasi saja yaitu posisi ATAS (TOP) tumpukan.
3. Ada beberapa macam *stack* yaitu *stack* dengan *array* dan *double stack* dengan *array*

Disetujui Aslab Alfina (2018041) Tgl:	Ttd / Paraf
---	-------------

## **BAB 4**

### **SORTING**

#### **4.1 Jumlah Pertemuan**

2 x 50 menit.

#### **4.2 Tujuan**

1. Praktikan dapat memahami pengertian dari *Sorting*
2. Praktikan mengetahui dan memahami cara kerja dari sistem *sorting*
3. Praktikan dapat mengoperasikan atau menerapkan metode *sorting*

#### **4.3 Alat dan Bahan**

1. Perangkat komputer
2. Perangkat lunak: Dev C++
3. Modul Struktur Data 2022

#### **4.4 Landasan Teori**

##### **A. Pengertian Sorting**

*Sorting* dalam arti bahasa adalah pengelompokan sebuah data yang tersusun secara acak yang kemudian di urutkan secara *ascending* (urutan naik) maupun *descending* (urutan turun). Pengurutan data dalam struktur data sangat penting terutama untuk data yang bertipe data numerik ataupun karakter.

Data Acak : 9 1 0 5 78 31 55 10

*Ascending* : 0 1 5 9 10 31 55 78

*Descending* : 78 55 31 10 9 5 1 0

Metode sorting terdiri dari :

- 1) *Bubble Sort*
- 2) *Exchange Sort*
- 3) *Insection sort*
- 4) *Selection Sort*
- 5) *Quicksort*

Deklarasi *array* dalam *sorting* bentuk umumnya adalah sebagai berikut.

```
int data[100]; // banyak data di sediakan
int a,b; // variable biasanya 2 atau lebih
```

Kemudian dalam *sorting* terdapat prosedur untuk menukar 2 buah data dengan bentuk umumnya sebagai berikut

```

void tukar
{
    int tmp = data[a];
    data[a] = data[b];
    data[b] = tmp;
}

```

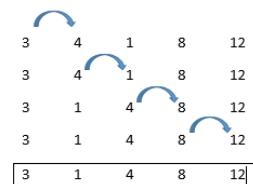
## B. Bubble Sort

*Bubble* adalah metode dalam *sorting* yang paling mudah logikanya. Diberi nama “*Bubble*” karena proses pengurutannya berangsur angsur bergerak/berpindah ke posisi yang tepat, seperti gelembung yang keluar dari dalam gelas bersoda. Cara kerja metode ini adalah dengan cara membandingkan elemen sekarang dengan elemen berikutnya. Metode ini seolah-olah menggeser satu elemen dari kanan ke kiri atau sebaliknya, tergantung jenis pengurutannya.

Contoh logika *Bubble Sort* :

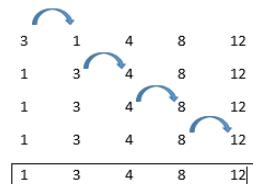
Data : 3,4,1,8,12

Proses 1



Gambar 4.1 Proses 1 *Bubble Sort*

Proses 2



Gambar 4.2 Proses 2 *Bubble Sort*

Proses akan berulang sebanyak 4 kali ( banyak data yang akan di sorting dikurangi 1 ).

### C. Exchange Sort

Exchange *Sort* adalah metode dalam *sorting* di mana dikatakan sangat mirip dengan metode *sorting bubble sort*. Namun, *exchange sort* membandingkan suatu elemen dengan elemen-elemen lainnya dalam *array* tersebut, dan melakukan pertukaran elemen jika perlu. Jadi ada elemen yang selalu menjadi elemen pusat (pivot). Sedangkan *Bubble sort* akan membandingkan elemen pertama/terakhir dengan elemen sebelumnya/sesudahnya, kemudian elemen tersebut itu akan menjadi pusat (pivot) untuk dibandingkan dengan elemen sebelumnya atau sesudahnya lagi, begitu seterusnya. Di situlah di mana letak perbedaan exchange dan *bubble sort* terletak.

Contoh logika Exchange *Sort* :

Data : 84,69,76,86,94,91

Proses 1

Pivot (Pusat)					
84	69	76	86	94	91
84	69	76	86	94	91
84	69	76	86	94	91
86	69	76	84	94	91
94	69	76	84	86	91
94	69	76	84	86	91

Gambar 4.3 Proses 1 Exchange *Sort*

Proses 2

Pivot (Pusat)					
94	69	76	84	86	91
94	76	69	84	86	91
94	84	69	76	86	91
94	86	69	76	84	91
94	91	69	76	84	86

Gambar 4.4 Proses 2 Exchange *Sort*

Proses 3

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

Gambar 4.5 Proses 3 Exchange Sort

Proses 4

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

Gambar 4.6 Proses 4 Exchange Sort

Proses 5

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

Gambar 4.7 Proses 5 Exchange Sort

Keterangan : Pivot adalah bagian yang di lingkari dan akan dibandingkan dengan data yang diberi anak panah.

#### D. Insertion Sort

*Insertion Sort* adalah metode dalam *sorting*. Di mana logikanya hampir mirip seperti mengurutkan sebuah kartu. Di mana kartu tersebut di urutkan selembar demi selembar. Dalam *insertion* pengurutan dimulai dari elemen-2 sampai elemen terakhir, jika ditemukan yang lebih kecil, maka akan ditempatkan pada posisi yang seharusnya.

Contoh logika *Insertion Sort* :

Data : 22,10,15,3,8,2

Proses 1

Proses 1						
0	1	2	3	4	5	
22	10	15	3	8	2	
Temp	Cek	Geser				
10	Temp<22?	Data ke-0 ke posisi 1				

Temp menempati posisi ke -0

0	1	2	3	4	5	
10	22	15	3	8	2	

Gambar 4.8 Proses 1 *Insertion Sort*

Proses 2

Proses 2						
0	1	2	3	4	5	
10	22	15	3	8	2	
Temp	Cek	Geser				
15	Temp<22	Data ke-1 ke posisi 2				
15	Temp>10	-				

Temp menempati posisi ke-1

0	1	2	3	4	5	
10	15	22	3	8	2	

Gambar 4.9 Proses 2 *Insertion Sort*

Proses 3

Proses 3						
0	1	2	3	4	5	
10	15	22	3	8	2	
Temp	Cek	Geser				
3	Temp<22	Data ke-2 ke posisi 3				
3	Temp<15	Data ke-1 ke posisi 2				
3	Temp<10	Data ke-0 ke posisi 1				

Temp menempati posisi ke-0

0	1	2	3	4	5	
3	10	15	22	8	2	

Gambar 4.10 Proses 3 *Insertion Sort*

#### Proses 4

Proses 4						
0	1	2	3	4	5	
3	10	15	22	8	2	
<b>Temp</b>   <b>Cek</b>   <b>Geser</b>						
8	Temp<22	Data ke-3 ke posisi 4				
8	Temp<15	Data ke-2 ke posisi 3				
8	Temp<10	Data ke-1 ke posisi 2				
8	Temp>3	-				
<b>Temp menempati posisi ke-1</b>						
0	1	2	3	4	5	
3	8	10	15	22	2	

Gambar 4.11 Proses 4 *Insertion Sort*

#### Proses 5

Proses 5						
0	1	2	3	4	5	
3	8	10	15	22	2	
<b>Temp</b>   <b>Cek</b>   <b>Geser</b>						
2	Temp<22	Data ke-4 ke posisi 5				
2	Temp<15	Data ke-3 ke posisi 4				
2	Temp<10	Data ke-2 ke posisi 3				
2	Temp<8	Data ke-1 ke posisi 2				
2	Temp<3	Data ke-0 ke posisi 1				
<b>Temp menempati posisi ke-0</b>						
0	1	2	3	4	5	
2	3	8	10	15	22	

Gambar 4.12 Proses 5 *Insertion Sort*

#### E. Selection Sort

*Selection* adalah metode dalam *sorting* pengurutan dengan cara elemen terkecil atau terbesar, sesuai permintaan. Untuk kemudian dibandingkan & ditukarkan dengan elemen data awal seterusnya sampai dengan seluruh elemen, sehingga akan menghasilkan pola data yang telah di *sort*.

Prinsip kerja *selection sort*:

- 1) Pengecekan dimulai data ke -1 sampai dengan data ke -n (misal i & pos).
- 2) Bandingkan data pos dengan data i+1 ( misal j ).
- 3) Jika data pos sesuai dengan kondisi maka pos = j dan j akan selalu bertambah ( j++ ) sampai melewati n.

- 4) Lakukan langkah 2 dan 3 untuk bilangan berikutnya sampai didapatkan urutan yang optimal.
- 5) Mengecek apakah  $i \neq pos$  jika ya, maka data akan di tukar.
- 6) Lakukan proses 1 sampai 5 sampai di dapatkan proses ke  $n-1$ .

Contoh logika *selection sort* :

Data : 32 75 69 58 21 40

Proses 1

0	1	2	3	4	5
32	75	69	58	21	40

<u>Pembanding</u>	<u>Posisi</u>
32 < 75	0
32 < 69	0
32 < 58	0
32 > 21 ( <u>tukar idx</u> )	4
21 < 40	4

Tukar data ke-0 (32) dengan data ke-4 (21)  

0	1	2	3	4	5
21	75	69	58	32	40

Gambar 4.13 Proses 1 *Selection Sort*

Proses 2

0	1	2	3	4	5
21	75	69	58	32	40

<u>Pembanding</u>	<u>Posisi</u>
75 > 69 ( <u>tukar idx</u> )	2
69 > 58 ( <u>tukar idx</u> )	3
58 > 32 ( <u>tukar idx</u> )	4
32 < 40	4

Tukar data ke-1 (75) dengan data ke-4 (32)  

0	1	2	3	4	5
21	32	69	58	75	40

Gambar 4.14 Proses 2 *Selection Sort*

### Proses 3

0	1	2	3	4	5
21	32	<b>69</b>	58	75	40

Pembanding	Posisi
69 > 58 (tukar idx)	3
58 < 75	3
58 > 40	5

Tukar data ke-2 (69) dengan data ke-5 (40)  

0	1	2	3	4	5
21	32	40	58	75	69

Gambar 4.15 Proses 3 Selection Sort

### Proses 4

0	1	2	3	4	5
21	32	40	<b>58</b>	75	69

Pembanding	Posisi
58 < 75	3
58 < 69	3

Tukar data ke-3 (58) dengan data ke-3 (58)  

0	1	2	3	4	5
21	32	40	58	75	69

Gambar 4.16 Proses 4 Selection Sort

### Proses 5

0	1	2	3	4	5
21	32	40	58	<b>75</b>	69

Pembanding	Posisi
75 > 69	5

Tukar data ke-4 (75) dengan data ke-5 (69)  

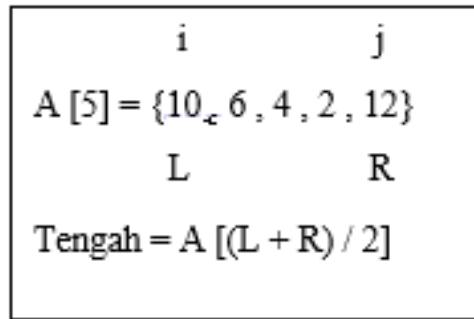
0	1	2	3	4	5
21	32	40	58	69	75

Gambar 4.17 Proses 5 Selection Sort

## F. Quick sort

*Quick sort* adalah metode dalam *sorting* yang mana adalah pengurutan membandingkan suatu elemen yang disebut pivot (memilih *index* tengah dari *array*) dengan elemen yang lain dan menyusunnya sedemikian rupa sehingga elemen-elemen lainnya yang lebih kecil daripada pivot tersebut terletak di sebelah kirinya dan elemen - elemen lain yang lebih besar daripada pivot terletak di sebelah kanannya.

Dengan demikian telah terbentuk dua *sublist*, lalu pada *sublist* kiri dan *sublist* kanan anggap sebuah *list* baru dan kerjakan proses yang sama seperti sebelumnya. Demikian seterusnya sampai tidak terdapat *sublist* lagi.



Gambar 4.18 Menentukan Nilai Tengah (*Quick Sort*)

Pada Gambar 6.1 ini merupakan cara untuk menentukan nilai tengah dari sebuah pivot atau *mid* pada data *array*, Setelah menentukan nilai tengah/pivot kemudian masuk ke dalam Algoritma Quicksort.

Algoritma *quick sort* :

- 1) Mengecek apakah  $A[i] < \text{tengah}$ 
  - a) Jika ya,  $i++$  ; Ke langkah 1
  - b) Jika tidak,  $i = i$  ; ke langkah 2
- 2) Mengecek apakah  $A[j] > \text{tengah}$ 
  - a) Jika ya,  $j--$  ; ke langkah 2
  - b) Jika tidak,  $j = j$  ; ke langkah 3
- 3) Mengecek apakah  $i \leq j$ 
  - a) Jika ya, tukar  $A[i]$  dengan  $A[j]$   
 $i++, j--$  ; ke langkah 4
  - b) Jika tidak, ke langkah 4
- 4) Mengecek apakah  $i < j$ 
  - a) Jika ya, ke langkah 1
  - b) Jika tidak, ke langkah 5
- 5) Mengecek apakah  $L < j$ 
  - a) Jika ya, panggil fungsi Quicksort(  $L, J$  )
  - b) Jika tidak, ke langkah 6

- 6) Mengecek apakah  $i < R$
- Jika ya, panggil Quicksort (  $i, R$  )
  - Jika tidak, Selesai.

#### 4.5 Tugas Praktikum ke-1: Program Buble Sort

*Source Code :*

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

int main () {

    int data;
    //input banyak data
    cout<<"======"<<endl;
    cout<<"Banyak data : ";
    cin>>data;
    cout<<"======"<<endl;
    int list [data];

    // perulangan inputan data
    for (int banyak=1;banyak<=data;banyak++) {
        cout<<"Masukkan data ke-"<<banyak<< " : ";
        cin>>list [banyak];
        list [data]=list [banyak];

    }
    //tampilan data yang inputkan
    cout<<"======"<<endl;
    cout<<"Data Awal : ";
    for (int d=1;d<=data;d++) {
        cout<<" "<<list [d];
    }
    cout<<endl;
    cout<<"======"<<endl;

    int temp;
    //proses perurutan
    cout<<"\n\n";
    for (int i=1;i<=data-1;i++) {
        cout<<"Proses ke "<<i<<" = ";
        for (int x=1;x<data;x++) {
            if (list [x]>list [x+1]){
                temp=list [x];
                list [x]=list [x+1];
                list [x+1]=temp;
            }
        }
        for (int y=1;y<=data;y++) {
            cout<<" "<<list [y];
        }
        cout<<endl;
    }
    cout<<endl;
    cout<<"Data yang diurutkan : ";
    for (int d=1;d<=data;d++) {
        cout<<" "<<list [d];
    }
}
```

```
    }
```

Tampilan program :

```
le_inputan && "/Users/arifin/Documents/DAT
kum S.Data/per4/*Bubble_inputan
=====
Banyak data : 5
=====
Masukkan data ke-1 : 12
Masukkan data ke-2 : 5
Masukkan data ke-3 : 8
Masukkan data ke-4 : 10
Masukkan data ke-5 : 2
=====
Data Awal : 12 5 8 10 2
=====

Proses ke 1 = 5 8 10 2 12
Proses ke 2 = 5 8 2 10 12
Proses ke 3 = 5 2 8 10 12
Proses ke 4 = 2 5 8 10 12
=====
Data yang diurutkan : 2 5 8 10 12
```

Gambar 4.19 Tampilan Program Buble Sort

Analisa Program :

Pada analisa program di atas di mana program tersebut melakukan proses program *buble sort* yang untuk malukan *sorting* secara *ascending* dengan *inputan* yang *user* tentukan. Pada saat program di jalankan user akan di minta untuk meng *input* banyak data yang akan nantinya sebagai banyak data dari *array*. Setelah menginputkan banyak data *array* maka selanjutnya akan melakukan perulangan sebanyak yang *user* tentukan dan *user* akan memasukkan data dari data ke 1 hingga yang *user* tentukan. Setelah di *input* kan maka akan di tampilkan lagi sebagai data awal sebelum melakukan *shorting* dengan menggunakan perulangan. Selanjutnya yaitu menuju ke baris 33 yang di mana akan melakukan proses perulangan. Yang pertama yaitu proses ke satu yang di mana pada baris ke 33 ber kondisi  $i \leq data - 1$  dan akan berlanjut ke baris 35 untuk melakukan perulangan dengan menentukan hasil *sorting* dari proses pertama. Selanjutnya yaitu menjalankan *if* yang di mana terdapat sebuah kondisi untuk melakukan perbandingan dari data ke satu dengan data ke dua dengan kondisi (*list [x] >= list [x+1]*). Jika data ke satu lebih besar atau sama dengan data ke dua maka akan menjalankan *statement* pada baris 37 sampai 39, jika ya maka data ke satu dengan data ke dua akan melakukan perpindahan dengan menggunakan *statement* yang ada pada *if*. Selanjutnya yaitu membandingkan data ke dua dengan data ke tiga , jika data ke dua lebih kecil dari ke tiga maka tidak akan menjalan *statement* pada *if* dan akan langsung lanjut ke data ke tiga

dengan data 4 yang akan dilakukan perbandingan begitu juga seterusnya setelah melakukan perbandingan maka akan di tampilkan sebagai hasil proses ke 1 dan berlanjut ke proses 2 dan seterusnya.

#### 4.6 Tugas Praktikum ke-2: Program Exchange Sort

*Source Code :*

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

int main (){
    int jumlah, pilih;
    int tanda ;
    cout<<"======"<<endl;
    cout<<"Banyak data : ";
    cin>>jumlah;
    cout<<"======"<<endl;
    int data [jumlah];
    for (int banyak=1;banyak<=jumlah;banyak++) {
        cout<<"Masukkan data ke-"<<banyak<< " : ";
        cin>>data [banyak];
        data [jumlah]=data [banyak];

    }
    cout<<endl;
    cout<<"Data awal : ";
    for (int x=1 ; x<=jumlah;x++) {
        cout<<data[x]<<" ";
    }

    do {
        cout<<endl;

cout<<"======"<<endl;
        cout<<"Sorting secara ? \n";
        cout<<"1. Ascending \n";
        cout<<"2. Descending \n";
        cout<<"3. Axit \n";

cout<<"======"<<endl;
        cout<<"Masukkan pilihan (1/2) : ";cin>>pilih;

cout<<"======"<<endl;
        switch (pilih)
        {
            case 1 :
                //melakukan pemprosesan asceding
exchange sort
                cout<<endl<<endl;
                for (int b=1;b<=jumlah-1;b++){
                    cout<< "Proses "<<b<< " = ";
                    //perulangan untuk mencari
hasil dari sebuah proses
                    for (int a=b;a<jumlah;a++){
                        //kondisi yang akan
melakukan perurutan perpidahan data dengan ascending
```

```

        if (data [b]>=data
[a+1]) {
            tanda =data[b];
            data
            [a+1]=tanda;
        }
    }
    //menampilkan hasil dari proses
    yang pertama hingga yang terakhir
    for (int a=1;a<=jumlah;a++) {
        cout<<data[a]<<" ";
    }
    cout<<endl;
}
cout<<endl;
//hasil dari pemprosesan yang terakhir
atau menampilkan hasil data dari hasil proses exchange
cout<<"Hasil shorting : ";
for (int x=1;x<=jumlah;x++) {
    cout<<" "<<data [x];
}

}
break;
case 2 :
    //melakukan pemprosesan desceding
exchange sort
    cout<<endl<<endl;
    for (int b=1;b<=jumlah-1;b++) {
        cout<< "Proses "<<b<<" = ";
        //perulangan untuk mencari
        hasil dari sebuah proses
        for (int a=b;a<=jumlah-
1;a++) {
            //kondisi yang akan
            melakukan perurutan perpidahan data dengan descending
            if (data [b]<=data
[a+1]) {
                tanda =data[b];
                data
                [a+1]=tanda;
            }
        }
    }
    //menampilkan hasil dari proses
    yang pertama hingga yang terakhir
    for (int a=1;a<=jumlah;a++) {
        cout<<data[a]<<" ";
    }
    cout<<endl;
}
//hasil dari pemprosesan yang terakhir
atau menampilkan hasil data dari hasil proses exchange
cout<<"Hasil shorting : ";
for (int x=1;x<=jumlah;x++) {
    cout<<" "<<data [x];
}

```

```

        }
        break;
    }

}
while (pilih!=3);
cout<<"Terima kasih"<<endl;
cout<<"-----"
"<<endl;
return 0;
}

```

Tampilan program :

```

change_input && "/Users/arifin/Documents/DA
tikum S.Data/per4/"exchange_input
=====
Banyak data : 4
=====
Masukkan data ke-1 : 60
Masukkan data ke-2 : 26
Masukkan data ke-3 : 45
Masukkan data ke-4 : 15

Data awal : 60 26 45 15
=====
Sorting secara ?
1. Ascending
2. Descending
3. Axit
=====
Masukkan pilihan (1/2) : 1
=====

Proses 1 = 15 60 45 26
Proses 2 = 15 26 60 45
Proses 3 = 15 26 45 60

Hasil shorting : 15 26 45 60
=====
```

Gambar 4.20 Tampilan Program Exchange Sort

Analisa Program :

Pada program di atas adalah program *exchange sort* dengan *ascending* dan *descending*. Pada saat di jalankan program tersebut user akan melakukan *inputan* banyak data yang akan dilakukan *shorting ascending* dan *descending*. Selanjutnya yaitu akan terjadi perulangan sebanyak data yang *user input* kan dari masing-masing data dari data ke satu hingga data yang terakhir. Saat *user* udah meng *input* kan data maka akan terjadi perulangan untuk menampilkan data-data yang *user input* kan sebagai data awal. Selanjutnya *user* akan diminta untuk melakukan pilihan apakah data *user* akan dilakukan *shorting ascending* atau *sorting descending* dengan memilih salah satu. Jika *user* memilih satu maka akan menjalankan *case 1* yang di mana *case* satu tersebut sebagai untuk

menjalan program *ascanding* yang di mana dalam *case* tersebut terdapat perulangan hasil proses. Pada *for* di baris 40 terdapat *inisialisasi*  $b=1$  dengan kondisi  $b \leq \text{jumlah}-1$  dengan melakukan *incerment*. Saat terjadi perulangan akan menampilkan proses ke 1 dan akan berlanjut ke *for* selanjutnya ke *for* 43 untuk mencari hasil dari sebuah proses ke satu. Pada *for* tersebut yaitu terdapat *inisialisasi*  $a=b$  yang di mana  $b=1$  dengan kondisi  $a < \text{jumlah}$  dan akan melakukan *increment*. Dan selanjutnya yaitu akan menjalankan *if* yang ada dalam *for* yang *if* tersebut akan melakukan perbandingan dengan data ke 1 dengan data ke 2 seperti pada gambar yang di mana akan melakukan perbandingan antara 60 dengan 26. Jika 60 lebih besar atau sama dengan 26 maka akan terjadi pertukaran data dengan menjalankan *statement* dalam *if* tersebut untuk melakukan pertukaran data. Selanjutnya yaitu menuju ke *for* pada baris 43 yang di mana  $a=b$  atau  $b=1$  dengan kondisi  $a < \text{jumlah}$  dan melakukan *increment* dan akan berlanjut perbandingan yang di mana data ke 1 dengan data ke 3 yaitu apakah data 26 lebih besar atau sama dengan 45 jika tidak maka akan berlanjut ke selanjutnya dan tidak akan menjalankan *statement* pada *if*. Dan akan berlanjut seterusnya sama kondisi bernilai *false* dan akan ditampilkan sebagai hasil proses ke 1 dan seterusnya. Jika *user* memilih 2 yaitu menjalankan *case* 2 yang di mana terdapat program *exchange* dengan *descending*. Selanjutnya yaitu akan menjalankan *for* pada baris 68 untuk menampilkan proses ke 1. Selanjutnya yaitu akan berlanjut ke *for* berikutnya yaitu pada baris 71 untuk mencari hasil dari proses ke 1 yang akan terjadi perulangan dan perbandingan. Pada *for* tersebut yang di mana  $a=b$  atau  $b=1$  dan terdapat sebuah kondisi  $a \leq \text{jumlah}-1$  dengan melakukan *increment*. Selanjutnya yaitu berlanjut ke *if* yang terdapat sebuah kondisi yang akan melakukan perbandingan yaitu apakah data ke satu lebih kecil dari data ke 2 jika lebih kecil maka akan menjalani *statement* yang ada pada *if* untuk melakukan pertukaran data. Jika sudah maka akan kembali ke *for* untuk perulangan yang di mana akan terjadi perbandingan data ke satu dengan ketiga, apakah data ke satu lebih kecil dari data ke 3 jika tidak maka tidak akan terjadi perulangan begitu juga seterusnya dan jika kondisi dari *for* tersebut *false* maka akan menampilkan sebagai hasil dari proses ke 1 dan seterusnya.

#### 4.7 Tugas Praktikum ke-4: Mengurutkan Dengan Insertion Sort dan Selection.

##### A. *Insertion*

Proses 1 :

Tabel 4.1 *Insertion* proses 1

0	1	2	3	4
52	31	40	18	25
temp	Cek	Geser		
31	Temp<52	Data ke-0 keposisi 1		
Temp menempati posisi ke-0				
0	1	2	3	4
31	52	40	18	25

Proses 2 :

Tabel 4.2 *Insertion* proses 2

0	1	2	3	4
31	52	40	18	25
temp	Cek	Geser		
40	Temp<52	Data ke-1 keposisi 2		
40	Temp<31	-		
Temp menempati posisi ke-1				
0	1	2	3	4
31	40	52	18	25

Proses 3 :

Tabel 4.3 *Insertion* proses 3

0	1	2	3	4
31	40	52	18	25
temp	Cek	Geser		
18	Temp<52	Data ke-2 keposisi 3		
18	Temp<40	Data ke-1 keposisi 2		
18	Temp<31	Data ke-0 keposisi 1		
Temp menempati posisi ke-0				
0	1	2	3	4
18	31	40	52	25

Proses 4 :

Tabel 4.4 *Insertion* proses 4

0	1	2	3	4
18	31	40	52	25
temp	Cek	Geser		
25	Temp<52	Data ke-3 keposisi 4		
25	Temp<40	Data ke-2 keposisi 3		
25	Temp<31	Data ke-1 keposisi 2		
25	Temp<18	0		

Temp menempati posisi ke-1

0      1      2      3      4  
18     25     31     40     52

B. Selection

Proses 1 :

Tabel 4.5 *Selection* proses 1

0	1	2	3	4
52	31	40	18	25
Perbandingan		Posisi		
52<31 (tukar index)		1		
31<40		1		
31<18 (tukar index)		3		
18<25		3		

Tukar data ke-0 (52) dengan data ke-3(18)

0      1      2      3      4  
18     31     40     52     25

Proses 2 :

Tabel 4.6 *Selection* proses 2

0	1	2	3	4
18	31	40	52	25
Perbandingan				Posisi
31<40				1
31<52				1
31<25				4
Tukar data ke-1 (31) dengan data ke-4(25)				
0	1	2	3	4
18	25	40	52	31

Proses 3 :

Tabel 4.7 *Selection* proses 3

0	1	2	3	4
18	25	40	52	31
Perbandingan				Posisi
40<52				2
40<31 (tukar index)				4
Tukar data ke-2 (40) dengan data ke-4(31)				
0	1	2	3	4
18	25	31	52	40

Proses 4 :

Tabel 4.8 *Selection* proses 4

0	1	2	3	4
18	25	31	52	40
Perbandingan				Posisi
52<40 (tukar index)				4
Tukar data ke-3 (52) dengan data ke-4(40)				
0	1	2	3	4
18	25	31	40	52

#### 4.8 Tugas Praktikum ke-3: Program Insertion Sort.

*Source Code :*

```
#include <iostream>
using namespace std;
int main (){
    int temp,x,j;
    int jumlah;
    cout<<endl;
    cout<<"Masukkan Jumlah data : ";cin>>jumlah;
    int data[jumlah];
    cout<<"=====\n";
    for (int x = 0; x < jumlah; x++)
    {
        cout<<"Data ke "<<x+1<<" = ";cin>>data[x];
    }
    cout<<"=====\n";
    cout<<"Data awal : ";
    for (int i= 0; i<jumlah;i++){
        cout<<data[i]<<" ";
    }
    cout<<endl;
    cout<<"=====\n";
    cout<<"Data yang telah diurutkan : "<<endl;
    cout<<"=====\n";
    cout<<"Pengurutan Ascending \n";
    cout<<"=====\n";
    for (x=1;x<jumlah;x++) {
        temp = data[x];
        for (j=x-1;j>=0;j--) {
            if (temp<=data[j]){
                data[j+1]=data[j];
                data[j]=temp;
                temp = data[j];
            }
        }
        cout<<"Proses "<<x<<" = ";
        for (int y=0; y<jumlah;y++) {
            cout<<data[y]<<" ";
        }
        cout<<endl;
    }
    cout<<"=====\n";
    cout<<"Pengurutan Descending \n";
    cout<<"=====\n";
    for (x=1;x<jumlah;x++) {
        temp = data[x];
        for (j=x-1;j>=0;j--) {
            if (temp>=data[j]){
                data[j+1]=data[j];
                data[j]=temp;
                temp = data[j];
            }
        }
        cout<<"Proses "<<x<<" = ";
        for (int y=0; y<jumlah;y++) {
            cout<<data[y]<<" ";
        }
    }
}
```

```

        cout<<endl;
    }
    cout<<"=====\n";
}

```

Tampilan program :

```

A KULIAH/semester 2/Praktikum 5
.Data/per5/"inputan_insertion

Masukkan Jumlah data : 5
=====
Data ke 1 = 41
Data ke 2 = 28
Data ke 3 = 17
Data ke 4 = 32
Data ke 5 = 50
=====
Data awal : 41 28 17 32 50
=====
Data yang telah diurutkan :
=====
Pengurutan Ascending
=====
Proses 1 = 28 41 17 32 50
Proses 2 = 17 28 41 32 50
Proses 3 = 17 28 32 41 50
Proses 4 = 17 28 32 41 50
=====
Pengurutan Descending
=====
Proses 1 = 28 17 32 41 50
Proses 2 = 32 28 17 41 50
Proses 3 = 41 32 28 17 50
Proses 4 = 50 41 32 28 17
=====
```

Gambar 4.21 Tampilan Program *Insertion Sort*

Analisa Program :

Pada analisa program di atas yaitu program *insertion sort*. Pada saat program dijalankan maka *user* akan diminta untuk meng *input* jumlah data yang ingin melakukan *insertion* dengan secara *ascending* maupun *descending*. Jika *user* sudah meng *inputkan* banyak data maka akan dilakukan sebuah perulangan yang di mana perulangan tersebut sebanyak jumlah yang *user* masukkan. Selanjutnya *user* akan memasukkan data dari masing-masing perulangan dan ditampilkan langsung sebagai data awal sebelum dilakukan *ascending* maupun *descending*. Jika data awal sudah ditetapkan maka akan berlanjut ke proses pengurutan dengan secara *ascending* dengan menggunakan *nested loop* atau *for* dalam *for*. Selanjutnya yaitu akan menjalankan *for* pada baris yang 28 yang di mana ber *inisialisasi* *x=1* dengan kondisi *x<jumlah* dan melakukan *increment*. Jika kondisi pada *for* tersebut *true* maka akan menjalankan *statement* yang ada di dalamnya yang di mana terdapat *temp=data [x]*; yang menentukan *inisialisasi* dari *index*. Selanjutnya yaitu menjalankan *for* yang kedua pada baris 30 yang di mana terdapat sebuah *inisialisasi* *j=x-1* dengan kondisi *j>=0* dan melakukan *decrement*. Jika kondisi *for* tersebut

bernilai *true* maka akan berlanjut ke *statement* di dalamnya yang terdapat if dengan kondisi *temp* $\leq$ *data[j]* untuk melakukan sebuah perbandingan yang di mana jika nilai dari *temp* tersebut lebih kecil maka akan menjalankan *statement* yang ada di dalamnya untuk melakukan perpindahan. Dan selanjutnya yaitu menuju ke *for* berikutnya pada baris 38 yang akan menampilkan hasil proses yang pertama jika dari *for* sebelumnya dengan kondisi *false* atau ditemukannya hasil dari proses 1. Dan ini akan dilakukan secara perulangan sampai proses yang terakhir atau dari kondisi *for* pada baris 28 bernilai *false*. Selanjutnya yaitu ke proses pengurutan secara *descending* yang akan menjalankan *for* pada baris yang 47 yang di mana ber *inisialisasi* *x=1* dengan kondisi *x* $\leq$ *jumlah* dan melakukan *increment*. Jika kondisi pada *for* tersebut *true* maka akan menjalankan *statement* yang ada di dalamnya yang di mana terdapat *temp* $=$ *data[x]*; yang menentukan *inisialisasi* dari *index*. Selanjutnya yaitu menjalankan *for* yang kedua pada baris 49 yang di mana terdapat sebuah *inisialisasi* *j=x-1* dengan kondisi *j* $\geq$  $0$  dan melakukan *decrement*. Jika kondisi *for* tersebut bernilai *true* maka akan berlanjut ke *statement* di dalamnya yang terdapat if dengan kondisi *temp* $\leq$ *data[j]* untuk melakukan sebuah perbandingan yang di mana jika nilai dari *temp* tersebut lebih besar maka akan menjalankan *statement* yang ada di dalamnya untuk melakukan perpindahan dan akan di tampilkan sebagai hasil dari proses pada *for* yang selanjutnya.

#### 4.9 Tugas Praktikum ke-5: Program Quick Sort.

*Source code :*

```
#include <iostream>
using namespace std;
int data[5] = {69, 89, 34, 56, 131 };

void QuickSort(int L, int R)
{
    int i, j;
    int mid;
    i = L;
    j = R;
    mid = data[(L + R) / 2];
    do
    {
        while (data[i] > mid)
            i++;
        while (data[j] < mid)
            j--;
        if (i <= j)
```

```

    {
        int t;
        t = data[j];
        data[j] = data[i];
        data[i] = t;
        i++;
        j--;
        cout << "Quicksort (" << L << ", " << R << ") =
    ";
        for (int i = 0; i < 5; i++)
        {
            cout << data[i] << " ";
        }
        cout << endl;
    }
} while (i < j);

if (L < j) QuickSort(L, j);
if (i < R) QuickSort(i, R);

}

int main(){
    cout << "Proses Quick Sort" << endl;
    cout << "-----" << endl;
    cout << "Data awal : ";
    for (int i = 0; i < 5; i++)
    {
        cout << data[i] << " ";
    }
    cout << endl;
    QuickSort(0, 4);
}

```

Tampilan program :

```

kum S.Data/per6/"quicksort_tugas
Proses Quick Sort
-----
Data awal : 69 89 34 56 131
Quicksort (0, 4) = 69 89 131 56 34
Quicksort (0, 3) = 131 89 69 56 34
Quicksort (0, 1) = 131 89 69 56 34
Quicksort (1, 3) = 131 89 69 56 34
Quicksort (3, 4) = 131 89 69 56 34
arifin@MacBook-Air-Arifin per6 %

```

Gambar 4.22 Tampilan hasil *Quicksort*

### Analisa Program :

Pada program di atas yaitu membuat program *quick sort* secara *descending* dengan banyak data 5 yang telah ditetapkan. pada saat program dijalankan yang dimana perulangan tersebut akan menampilkan data awal yang akan dilakukan *sorting* secara *descending* pada baris yang ke 45-48. Selanjutnya jika perulangan tersebut sudah bernilai *false* atau sudah terpenuhi maka akan berlanjut ke baris 50 yang melakukan pemanggilan *quicksort* yang sudah ditetapkan nilai variabel secara langsung. Selanjutnya yaitu akan menjalankan *void* yang sudah di panggil yang di mana terdapat *inisialisasi*  $i=L$  atau dengan kata lain  $L=0$  dan  $j=R$  yaitu  $R=4$ . Selanjutnya yaitu baris yang ke 11 untuk mencari *index tengah* dengan cara  $mid=data[(L + R) / 2]$ , jika *index tengah* sudah di temukan maka akan berlanjut ke baris 12 yang di mana terdapat perulangan *do while* yang *dimana* dalam *statement* do whlie tersebut juga terdapat sebuah kondisi *while* yaitu  $data[i] > mid$  jika ya maka akan dilakukan *increment* dan akan berlanjut ke baris. Selanjutnya yaitu *while* dengan kondisi  $data[j] < mid$  jika ya maka akan terjadi *decrement*. Selanjutnya yaitu menjalankan program pada baris yang ke 19 yang di mana terdapat *if* dengan kondisi  $(i \leq j)$  jika iya maka akan menjalankan *statement* yang ada di dalamnya. Di dalam *statement* pada *if* tersebut akan terjadi pertukaran data yang di mana nilai variabel  $t = data[j]$  dan  $data[j] = data[i]$  dan  $data[i] = t$  dan variabel  $i$  dilakukan *increment* dan variabel  $j$  dilakukan *decrement*. Selanjutnya yaitu menjalankan program pada baris yang ke 27-32 yang di mana akan menampilkan data dari hasil *quicksort* ( $L, R$ ) dengan perulangan *for*. Selanjutnya yaitu menjalankan pada bari yang 34 *while* dengan kondisi  $(i < j)$  jika ya maka akan terjadi perulangan dengan proses yang sebelumnya jika tidak makan akan melanjutkan program di bawah yang di mana *if* dengan kondisi  $(L < j)$  jika ya maka akan memanggil fungsi *quicksort* ( $L, j$ ). dan berikutnya *if* dengan kondisi  $(i < R)$  jika ya maka akan memanggil fungsi *quicksort* ( $i, R$ ).

#### 4.10 Kesimpulan

1. *Sorting* dalam arti bahasa adalah pengelompokan sebuah data yang tersusun secara acak yang kemudian di urutkan secara *ascending* (urutan naik) maupun *descending* (urutan turun).
2. *Bubble* adalah metode dalam *sorting* yang paling mudah logikanya.
3. *Exchange Sort* adalah metode dalam *sorting* di mana dikatakan sangat mirip dengan metode *sorting buble sort*.
4. *Insertion Sort* adalah metode dalam *sorting*. Di mana logikanya hampir mirip seperti mengurutkan sebuah kartu. Di mana kartu tersebut di urutkan selembar demi selembar.
5. *Selection* adalah metode dalam *sorting* pengurutan dengan cara elemen terkecil atau terbesar, sesuai permintaan.
6. *Insertion Sort* merupakan sebuah teknik pengurutan dengan cara membandingkan dan mengurutkan dua data pertama pada *array*, kemudian membandingkan data para *array* berikutnya apakah sudah berada di tempat semestinya.
7. *Quicksort* adalah metode dalam *sorting* yang mana adalah pengurutan membandingkan suatu elemen yang disebut pivot.

Disetujui Aslab Alfina (2018041) Tgl:	Ttd / Paraf
---	-------------

## **BAB 5**

### **SEARCHING**

#### **5.1 Jumlah Pertemuan**

2 x 50 menit.

#### **5.2 Tujuan**

1. Praktikan dapat memahami pengertian dari *Searching*
2. Agar praktikan mengetahui dan memahami cara kerja dari sistem *Searching*
3. Praktikan dapat mengoperasikan atau menerapkan metode *Searching*

#### **5.3 Alat dan Bahan**

1. Perangkat komputer
2. Perangkat lunak: Dev C++
3. Modul Struktur Data 2022

#### **5.4 Landasan Teori**

##### **A. Pengertian Searching**

Algoritma pencarian (*searching algorithma*) adalah algoritma yang menerima sebuah argumen kunci dan dengan langkah-langkah tertentu akan mencari rekaman dengan kunci tersebut. Setelah proses pencarian dilaksanakan, akan diperoleh salah satu dari dua kemungkinan, yaitu data yang dicari berhasil ditemukan (*successful*) atau data yang kita cari tersebut tidak berhasil ditemukan (*unsuccessful*).

Ada beberapa macam teknik pencarian seperti pencarian *sekuensial* dan pencarian biner. Perbedaan dari dua teknik ini terletak pada keadaan data. Pencarian *sekuensial* digunakan apabila data dalam keadaan acak atau tidak terurut (contoh: *sequential search*). Sebaliknya, pencarian biner digunakan pada data yang sudah dalam keadaan urut (contoh: *Binary search* dan *interpolation search*).

##### **B. Sequential Searching (Pencarian Berurutan)**

Pencarian berurutan sering disebut pencarian linear merupakan metode pencarian yang paling sederhana. Pencarian berurutan menggunakan prinsip sebagai berikut : data yang ada akan dibandingkan satu per satu secara berurutan dengan data yang dicari sampai data tersebut ditemukan atau tidak ditemukan.

Pada dasarnya, pencarian ini hanya melakukan pengulangan dari 1 sampai dengan jumlah data. Pada setiap pengulangan, setiap data akan dibandingkan dengan yang dicari. Apabila sama, berarti data telah ditemukan, sebaliknya apabila sampai akhir pengulangan tidak ada data yang sama, berarti data tidak ditemukan. Pada kasus yang paling buruk, apabila terdapat n data maka harus dilakukan pencarian sebanyak n kali pula

Algoritma pencarian Berurutan dapat dituliskan sebagai berikut :

- 1)  $i \leftarrow 0$
- 2) Jika ( $\text{data}[i] = x$ ) maka data ditemukan , melanjutkan ke langkah 3
- 3) Selama ( $i \leq n$ ) kerjakan langkah ke 4
- 4)  $i = i + 1$  , melanjutkan langkah ke 5
- 5) Kembali ke langkah 2

Konsep : membandingkan setiap elemen satu per satu secara urut dan beruntun, mulai dari elemen pertama sampai dengan elemen yang terakhir. Ada 2 macam pencarian beruntun, yaitu pencarian pada *array* yang sudah terurut, dan pencarian pada *array* yang belum terurut.

Algoritma *sequential search* juga dapat menggunakan *break* untuk memberhentikan program saat data pertama kali ditemukan. Berikut algoritmanya :

- 1)  $pos \leftarrow 0$
- 2) Selama ( $pos \leq n$ ) kerjakan langkah ke 3
- 3) Cek jika ( $\text{data}[pos] = \text{cari}$ ) maka  $flag = 1$  kemudian berhenti yang berarti data ditemukan, jika tidak  $flag$  tetap & melanjutkan ke langkah 4
- 4)  $pos = pos + 1$  , melanjutkan langkah ke 5
- 5) Kembali ke langkah 2

### C. Binary Search

Salah satu syarat agar pencarian biner dapat dilakukan adalah data yang tersedia sudah dalam keadaan urut. Dengan kata lain, apabila data belum dalam keadaan urut, pencarian biner tidak dapat dilakukan. Dalam kehidupan sehari-hari, sebenarnya kita juga sering menggunakan pencarian biner. Misalnya saat ingin mencari suatu kata dalam kamus.

Algoritma dari pencarian biner dapat dijelaskan sebagai berikut :

*Inisialisasi awal :*

Kiri = 0, kanan = n ( index terakhir dari array ), *flag* = 0

- 1) Mengecek apakah ( kiri <= kanan && *flag* 0 )

Jika ya, ke langkah 2

Jika tidak, ke langkah 6.

- 2) Menghitung nilai tengah = ( kiri + kanan ) / 2

- 3) Mengecek apakah cari == data [ tengah ]

Jika ya, *flag* = 1. Ke langkah 6

Jika tidak, ke langkah 4

- 4) Mengecek apakah cari < data[tengah ]

Jika ya, kanan = tengah – 1. Ke langkah 1

Jika tidak ke langkah 5

- 5) Mengecek apakah cari > data [ tengah ]

Jika ya, kiri = tengah + 1. Ke langkah 1

Jika tidak, ke langkah 1

- 6) Mengecek apakah *flag* == 1

Jika ya, data di temukan

Jika tidak, data tidak di temukan

#### D. Interpolation Search

Teknik ini dilakukan pada data yang sudah terurut berdasarkan kunci tertentu. Teknik *searching* ini dilakukan dengan perkiraan letak data. Contoh ilustrasi : jika kita hendak mencari suatu kata dalam buku telepon, misal yang berawalan dengan huruf J, maka kita tidak akan mencarinya dari awal buku, tetapi kita langsung membukanya pada 1/3 atau 1/4 dari tebal buku tersebut. Rumus posisi relatif pencarian dihitung dengan rumus

$$\text{Posisi} = \frac{\text{cari} - \text{data}[low]}{\text{data}[high] - \text{data}[low]} \times (\text{high} - \text{low}) + \text{low}$$

Algoritma *Interpolation search* :

- 1) Mengecek apakah ( cari >= data[low] && cari <= data[high])

Jika ya, ke langkah 2

Jika tidak, ke langkah 6

- 2) Mencari nilai posisi =  $\frac{\text{cari}-\text{data}[\text{low}]}{\text{data}[\text{high}]-\text{data}[\text{low}]} \times (\text{high} - \text{low}) + \text{low}$

- pos = pembulatan ke bawah dari posisi
- 3) Mengecek apakah ( data [ pos ] == cari )  
 Jika ya, *flag* = 1. Ke langkah 6  
 Jika tidak, ke langkah 4
  - 4) Mengecek apakah ( data [ pos ] > cari )  
 Jika ya, *high* = pos – 1 . ke langkah 1  
 Jika tidak, ke langkah 5
  - 5) Mengecek apakah ( data [ pos ] < cari )  
 Jika ya, *low* = pos + 1. Ke langkah 1  
 Jika tidak, ke langkah 1.
  - 6) Mengecek apakah ( *flag* == 1 )  
 Jika ya, data ditemukan  
 Jika tidak, data tidak di temukan.

## 5.5 Tugas Praktikum ke-1: Program Sequential Searching

*Source code :*

```
#include <iostream>
using namespace std;
int main ()
{
    int cari,banyak,tanda;
    cout<<"=====\n";
    cout<<" Program sequential Searching \n";
    cout<<"=====\n";
    //proses inputan array
    cout<<"Masukkan banyak data : ";
    cin>banyak;
    cout<<"-----\n";
    int data[banyak];//index array
    //perulangan memasukan banyak data
    for (int i = 0; i <banyak; i++)
    {
        cout<<"Masukkan data ke-<<i+1<< " : ";
        cin>>data [i];
        data[banyak]=data[i];
    }
    cout<<"=====\n";
    int flag = 0; //inisialisasi flag
    //proses menampilkan data yang sudah diinputkan
    cout<<"Data awal : ";
    for(int a=0;a<banyak;a++)
    {
        cout<<data[a]<<" ";
    }
    cout<<endl;
    cout<<"=====\n";
    //proses data yang ingin di cari
```

```

cout<<"Masukkan data yang ingin di cari : ";cin>>cari;
for(int x=1;x<=banyak;//perulangan cek data di cari
{
    if(data[x]== cari)
    {
        flag = 1;
        break;
    }
}
if (flag==1)//kondisi pencarian data yang sudah di temukan
{
    cout<<"-----\n";
    cout<<"Data di temukan!\n";
    cout<<"-----\n";
}
else //jika data tidak di temukan
{
    cout<<"-----\n";
    cout<<" Data tidak temukan!\n";
    cout<<"-----\n";
}
}
}

```

Tampilan program :

```

/semester 2/Praktikum S.Data/per7/"tugas
=====
+-----+
 Program sequential Searching
+-----+
Masukkan banyak data : 6
-----
Masukkan data ke-1 : 45
Masukkan data ke-2 : 23
Masukkan data ke-3 : 1
Masukkan data ke-4 : 6
Masukkan data ke-5 : 7
Masukkan data ke-6 : 131
+=====+
Data awal : 45 23 1 6 7 131
+=====+
Masukkan data yang ingin di cari : 131
-----
Data di temukan!
=====
arifin@MacBook-Air-Arifin per7 %

```

Gambar 5.1 Tampilan *sequential searching*

Analisa Program :

Pada analisa program di atas yaitu program *sequential searching* yang menggunakan beberapa variabel untuk membuat program tersebut salah satunya yaitu *cari*, *banyak*, *tanda* dan *flag=0*. Dan pada saat program di jalankan maka *user* akan meng *input* banyak data dengan keinginannya sendiri. Dan selanjutnya yaitu pada baris yang ke 13 inputan banyak data yang sudah dimasukkan akan di cantumkan sebagai nilai dari variabel *array*. Dan selanjutnya yaitu menuju ke perulangan *for* dengan kondisi *i<banyak* dengan kata lain yaitu melakukan perulangan sebanyak data yang di *inputkan* dan setiap perulangan *user* akan memasukkan data ke 1 hingga banyak data yang di

inputkan. Jika sudah meng inputkan maka akan menjalankan baris yang ke 21 hingga 26 yaitu menampilkan data yang sudah dimasukan sebagai data awal. Selanjutnya yaitu untuk mencari dat dengan menggunakan *for* pada baris 33. Jika nilai kondisi pada *for* bernilai *true* maka akan berlanjut ke *statement* ada pada dalamnya yaitu *if*. Di dalam *if* terdapat kondisi untuk memperbandingkan apakah data itu sama jika tidak maka akan terjadi perulangan terus menurus sampai kondisi *for* bernilai *false* dan jika masih tidak di temukan maka akan menjalankan *else* yaitu data tidak di temukan tapi jika da di temukan maka *flag*=1 dan akan menampilkan data di temukan program selesai.

## 5.6 Tugas Praktikum ke-2: Program Binary Searching

*Source code :*

```
#include <iostream>
using namespace std;
int main()
{
    int banyak,cari,tengah,tanda;
    cout<<"=====\n";
    cout<<" PROGRAM BINARY SEARCHING \n";
    cout<<"=====+\n\n";
    //proses user input banyak index
    cout<<"Masukkan banyak index array: ";
    cin>>banyak;
    cout<<"-----\n";
    int data[banyak]; //menyimpan index yang sudah dimasukkan
    //perulangan for untuk memasukkan data yag sesuai dari
    index yang di tentukan
    for (int i = 0; i <= banyak; i++)
    {
        cout<<"Masukkan data ke-<<i<< " : ";
        cin>>data [i];
        data[banyak]=data[i];//menyimpan data yang sudah
di input
    }
    cout<<"Data awal : ";
    //perulangan untuk memproses data yang sudah diinput
    sebagai data awal
    for(int a=0; a<=banyak;a++)
    {
        cout<<data[a]<< " ";
    }
    //proses perurutan dengan menggunakan buble sort
    for (int b=0;b<banyak;b++) {
        for (int a=b;a<banyak;a++) {
            if (data [b]>=data [a+1]){
                // proses perpindahan data
                tanda =data[b];
                data [b]=data[a+1];
                data [a+1]=tanda;
            }
        }
    }
}
```

```

}
cout<<endl;
//proses menampilkan data yang sudah diurutkan
cout<<"Data yang diurutkan : ";
for (int x=0;x<=banyak;x++){
    cout<<" "<<data [x];

}
int kiri=0, kanan;
kanan=banyak;
int flag = 0;
cout<<"\nmasukan data yang dicari : ";
cin>>cari;
while (kiri <= kanan && flag == 0)//mengecek data
index array
{
    tengah = (kiri + kanan)/2; //mencari nilai
tengah
    if ( cari == data [ tengah ] )//mengecek data
    {
        flag = 1;
        break;
    }
    else if (cari < data[tengah])
    {
        kanan= tengah - 1;
    }
    else if (cari > data[tengah] )
    {
        kiri = tengah + 1;
    }
}
if ( flag == 1 )//kondisi jika data sudah di temukan
{
    cout<<"\n=====\\n";
    cout<<"Data di temukan!\\n";
    cout<<"=====\\n";
}
else
{
    cout<<"\\n=====\\n";
    cout<<"Data tidak temukan!\\n";
    cout<<"=====\\n";
}
}
}

```

Tampilan program :

```

emester 2/Praktikum S.Data/per7/"tugas_binary_se
arch
+=====+
 PROGRAM BINARY SEARCHING
+=====+
Masukkan banyak index array: 6
-----
Masukkan data ke-0 : 131
Masukkan data ke-1 : 45
Masukkan data ke-2 : 77
Masukkan data ke-3 : 32
Masukkan data ke-4 : 67
Masukkan data ke-5 : 1
Masukkan data ke-6 : 98
Data awal : 131 45 77 32 67 1 98
Data yang diurutkan : 1 32 45 67 77 98 131
masukan data yang dicari : 131

=====
Data di temukan!
=====
arifin@Air-Arifin per7 %

```

Gambar 5.2 Tampilan hasil *binary searching*

#### Analisa Program :

Pada analisa program di atas yaitu program *binary searching* yang terdapat beberapa variabel yaitu salah satunya variabel banyak, cari, tengah, tanda. Dan pada saat dijalankan maka *user* akan menginputkan banyak *array* sesuai keinginan dan itu akan dimasukkan ke dalam variabel *array* yaitu *data[banyak]*. Selanjutnya jika sudah menginputkan banyak *array* maka akan berlanjut pada baris yang ke 15 yang akan melakukan perulangan untuk menginputkan data dari setiap perulangan terjadi sampai kondisi pada perulangan tersebut bernilai *false* atau sebanyak yang diinputkan sebelumnya. Jika perulangannya terjadi maka akan menjalankan *statement* yang ada dalam *for* melakukan inputan dan setiap inputan akan dimasukkan ke dalam *array* yaitu *data[banyak]=data[i]*. Jika sudah menginputkan maka akan berlanjut untuk menjalankan program pada baris yang ke 23 yang menggunakan *for* untuk menampilkan data yang sudah diinputkan sebagai data awal. Karena data awal belum terurut dan tidak sesuai maka akan diurutkan menggunakan proses *bubble sort* untuk meng *sorting* perurutan secara *ascending* pada baris yang ke 28-38. Jika *for* pada baris yang ke kondisi masih bernilai *true* maka akan menjalankan *statment* yang ada di dalamnya selanjutnya yaitu *for* yang kedua jika *for* tersebut juga kondisinya bernilai *true* maka akan melanjutkan *statement* yang ada di dalamnya dan akan melakukan perbandingan data jika perbandingan tersebut bernilai *true* maka akan melakukan perpindahan data sesuai ketentuan atau operator tentukan. Jika proses perurutan sudah selesai maka akan melanjutnya

program selanjutnya yaitu untuk menampilkan data yang sudah di urutkan menggunakan *for* pada baris yang 41. Sebelum kita berlanjut kita ketemu beberapa variabel pada baris 46-48 yaitu *kiri=0*, *kanan=banyak*, *flag=0*. Selanjutnya jika sudah di tampilkan maka akan melanjutkan program selanjutnya yaitu untuk mencari data yang di inginkan dan *user* akan melakukan penginputan pencarian. Jika sudah di tampilkan sebagai data awal maka akan di urutkan terlebih dahulu sebagai syarat untuk algoritma pada *binary search*. Untuk proses perurutannya yaitu menggunakan proses *buble sort* pada baris 28 sampai 38 dengan ketentuan penjelasan materi sebelumnya. Dan selanjutnya yaitu jika sudah di urutkan maka akan di tampilkan dengan menggunakan *for* pada baris yang ke 41. Jika sudah di urutkan maka selanjutnya yaitu menuju ke variabel yang di mana terdapat *kiri =0* dan *kanan=banyak* dan *flag=0*. Selanjutnya yaitu menjalankan program ke 49-50 untuk menjalankan *inputan* yang ingin di cari. Jika sudah meng inputkan maka akan menjalan program pada bari 51 hingga 67 yaitu *while* dengan kondisi (*kiri<=kanan && flag==0*) jika hasilnya true maka akan akan mencari titik tengah terlebih dahulu dengan operator (*tengah=(kiri+kanan)/2*) dan jika sudah di temukan maka akan menjalankan *statement* yang ada di dalamnya yang mencari data yang diinginkan . Selanjutnya yaitu menjalankan program pada baris 54 yaitu *if* dengan kondisi *if(cari==data[tengah])* jika ya maka akan menjalankan *statement* yang di dalamnya maka *flag=1* dan program berhenti dan akan menampilkan baris yang 68 yang kondisinya *if(flag ==1)* dan akan menampilkan dan di temukan tapi jika maka akan menampilkan data di temukan. Selanjutnya yaitu baris yang ke 59 dengan kondisi *if (cari <data[tengah])* jika ya maka akan menjalan *statement* di dalamnya. Jika *statement* di dalamnya suda selesai maka akan kembali ke *whlie*, jika *while* bernilai false maka akan menampilkan data tidak di temukan dan program selesai.

## 5.7 Tugas Praktikum ke-3: Program Interpolation Searching

*Source code :*

```
#include <iostream>
using namespace std;
//mencari posisi array data ketika ditemukan
int interpolationSearch(int data[], int n, int cari)
```

```

{
    // Mencari ujung indeks
    int lo = 0, hi = n;
    while (lo <= hi && cari >= data[lo] && cari <= data[hi])
    {
        if (lo == hi)
        {
            if (data[lo] == cari) return lo;
            return -1;
        }
        // Mengira posisi dengan rumus
        int pos = lo + (((double)(hi - lo) /
        (data[hi] - data[lo])) * (cari - data[lo]));
        // Jika data cari ditemukan
        if (data[pos] == cari)
            return pos;
        // Jika cari lebih besar dari nilai indeks
        if (data[pos] < cari)
            lo = pos + 1;
        // Jika cari lebih kecil dari nilai indeks
        else
            hi = pos - 1;
    }
    return -1;
}
int main()
{
    int cari, banyak;
    cout<<"=====\n";
    cout<<" PROGRAM INTERPOLATION SEARCHING \n";
    cout<<"=====\n\n";
    cout<<"Masukkan banyak index array : ";
    cin>>banyak;
    int data[banyak];
    cout<<"-----\n";
    for (int i = 0; i <= banyak; i++)
    {
        cout<<"Masukkan data ke-"<<i<< " : ";
        cin>>data [i];
        data[banyak]=data[i];
    }
    cout << "Data awal: " << endl;
    for (int x = 0; x <= banyak; x++)
    {
        cout << " " << data[x];
    }
    cout<<endl;
    int list;
    //proses perurutan dengan bubble sort
    for (int b=0;b<banyak;b++) {
        for (int a=b;a<banyak;a++) {
            if (data [b]>=data [a+1]) {
                list =data[b];
                data [b]=data[a+1];
                data [a+1]=list;
            }
        }
    }
}

```

```

        cout<<"-----\n";
        cout<<endl;
        cout<<"Data yang diurutkan : ";
        for (int x=0;x<=banyak;x++) {
            cout<<" "<<data [x];

        }
        cout<<endl;
        int n = sizeof(data)/sizeof(data[0]); //menentukan ukuran
array(banyak data);
        cout << "======" << endl;
        cout << "Masukkan Data Yang Ingin Dicari : "; cin >>
cari;
        int tanda = interpolationSearch(data, n, cari);
        if (tanda != -1)
        {
            cout << ":: Data Ditemukan ::" << endl;
            cout << " Pada array ke = "<<tanda;
        }
        else
        {
            cout << ":: Data Tidak Ditemukan :: " << endl;
        }
    }
}

```

Tampilan program :

```

ester 2/Praktikum S.Data/per7/"tempCodeRunnerFile
=====
PROGRAM INTERPOLATION SEARCHING
=====

Masukkan banyak index array : 6
-----
Masukkan data ke-0 : 34
Masukkan data ke-1 : 131
Masukkan data ke-2 : 455
Masukkan data ke-3 : 123
Masukkan data ke-4 : 1
Masukkan data ke-5 : 5
Masukkan data ke-6 : 2
Data awal:
34 131 455 123 1 5 2
-----

Data yang diurutkan : 1 2 5 34 123 131 455
=====
Masukkan Data Yang Ingin Dicari : 131
:: Data Ditemukan :::
Pada array ke = 5
arifin@Air-Arifin per7 %

```

Gambar 5.3 Tampilan hasil *interpolation search*

Analisa Program :

Pada analisa program di atas yaitu program *interpolation* yaitu menggunakan parameter. Saat program di jalankan maka *user* akan menginputkan banyak *index* pada array dan akan di masukkan ke dalam *int data [banyak]*. Jika sudah di inputkan maka akan terjadi perulangan untuk menginputkan suatu data sebanyak yang sudah di tentukan oleh *user* sebelumnya. Di dalam *for* tersebut saat terjadi perulangan maka *user* akan

menginput data ke satu hingga data terakhir dan jika di inputkan maka akan disimpan ke dalam *array* sebagai data dengan menetapkan  $\text{data}[\text{banyak}] = \text{data}[\text{i}]$ . Jika sudah maka selanjutnya akan menampilkan data yang sudah di inputkan sebagai data awal menggunakan *for*. Selanjutnya jika sudah ditampilkan maka melakukan perurutan data yang sebelumnya *user* inputkan secara *ascending* dengan menggunakan *buble sort* seperti pada materi sebelumnya. Jika proses perurutan sudah selesai maka akan berlanjut untuk menampilkan hasil dari yang di urutkan dengan menggunakan *for*. Selanjutnya yaitu menjalankan program pada baris yang ke 72 yaitu  $\text{int n} = \text{sizeof}(\text{data})/\text{sizeof}(\text{data}[0])$  untuk menentukan banyak data. Jika sudah maka selanjutnya yaitu untuk mencari data sesuai *user input* kan pada baris yang ke 74. Selanjutnya yaitu menjalankan program pada baris yang ke 75 yaitu  $\text{int tanda} = \text{interpolationSearch}(\text{data}, \text{n}, \text{cari})$ ; yang terdapat pemanggilan *interpolation* untuk melakukan pencarian data. Pada parameter tersebut terdapat variabel  $\text{lo}=0$ ,  $\text{hi}=\text{n}$ . dan akan berlanjut ke *while* yang di mana terdapat *while* dengan kondisi *while* ( $\text{lo} \leq \text{hi} \&\& \text{cari} \geq \text{data}[\text{lo}] \&\& \text{cari} \leq \text{data}[\text{hi}]$ ). Jika hasil kondisi tersebut bernilai *false* maka akan menjalakan *statement* yang ada di dalamnya. *Statement* yang dalam *while* tersebut terdapat *if* dengan kondisi *if*( $\text{lo}==\text{hi}$ ) jika *true* maka akan menjalankan *statement* yang di dalamnya. Selanjutnya yaitu untuk mencari pos dengan operator  $\text{int pos} = \text{lo} + (((\text{double})(\text{hi} - \text{lo}) / (\text{data}[\text{hi}] - \text{data}[\text{lo}])) * (\text{cari} - \text{data}[\text{lo}]))$ ; jika sudah di temukan maka akan berlanjut di program berikutnya yaitu terdapat *if*  $y=\text{dengan}$  kondisi *if*( $\text{data}[\text{pos}] == \text{cari}$ ) jika *true* makan akan *return pos*. selanjutnya yaitu akan menjalankan program yang berikutnya yaitu *if* dengan kondisi *if* ( $\text{data}[\text{pos}] < \text{cari}$ ) jika iya maka  $\text{lo} = \text{pos}+1$ ; dan jika tidak  $\text{hi}=\text{pos}-1$ . Dan terjadi perulangan seterusnya sampai *while* tersebut bernilai *false* jika sudah maka akan *return -1*. Jika sudah temukan maka akan tanda  $= \text{interpolationSearch}(\text{data}, \text{n}, \text{cari})$  dan akan melakukan pengecekan pada *if* (*tanda* $\neq -1$ ), jika ya maka data di temukan jika tidak data tidak di temukan.

## 5.8 Kesimpulan

1. Algoritma pencarian (*searching algorithma*) adalah algoritma yang menerima sebuah argumen kunci dan dengan langkah-langkah tertentu akan mencari rekaman dengan kunci tersebut.
2. Syarat agar pencarian biner dapat dilakukan adalah data yang tersedia sudah dalam keadaan urut. Dengan kata lain, apabila data belum dalam keadaan urut, pencarian biner tidak dapat dilakukan.
3. Teknik *searching* ini dilakukan dengan perkiraan letak data.

Disetujui Aslab Alfina (2018041) Tgl:	Ttd / Paraf
---	-------------

## BAB 6

### LINKED LIST

#### 6.1 Jumlah Pertemuan

2 x 50 menit.

#### 6.2 Tujuan

1. Praktikan dapat memahami pengertian dari *Linked List*
2. Agar praktikan mengetahui dan memahami cara kerja dari sistem *Linked list* dan *Double linked list*
3. Praktikan dapat mengoperasikan atau menerapkan metode *Linked list*

#### 6.3 Alat dan Bahan

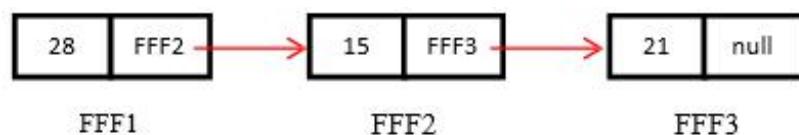
1. Perangkat komputer
2. Perangkat lunak: Dev C++
3. Modul Struktur Data 2022

#### 6.4 Landasan Teori

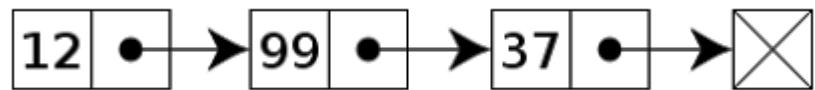
##### A. Linked List

###### 1) Pengertian Linked List

*Linked list* adalah sekumpulan elemen dari data yang bertipe sama yang saling terurut dan terhubung dengan bantuan variabel *pointer*, yang setiap data di dalamnya disebut *node* ( simpul ) menempati alokasi memori secara dinamis dan biasanya berupa *struct* yang terdiri dari beberapa *field*. *Linked list* juga merupakan suatu cara untuk menyimpan data dengan struktur sehingga dapat secara otomatis menciptakan suatu tempat baru untuk menyimpan data yang diperlukan. Berikut contohnya pada komputer serta ilustrasinya:



Gambar 6.1 *Linked List* pada *computer*



Gambar 6.2 Ilustrasi *Linked List*

Penjelasan :

- Node* pertama dengan data 12 terhubung dengan *node* kedua dengan data 99, dan seterusnya.
  - Anak panah pada kotak di sebelah kanan menunjukkan alamat dari *node* setelahnya
- 2) Perbandingan *Array* Dengan *Linked List*

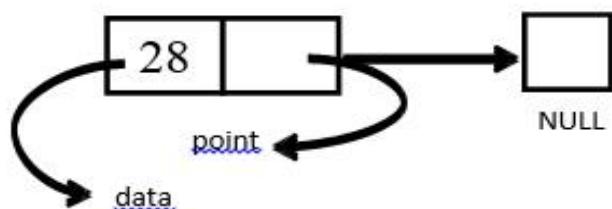
Tabel 6.1 Perbedaan *Linked List* dengan *Array*

Array	Linked List
Statis	Dinamis
Penambahan/penghapusan data terbatas	Penambahan/penghapusan data tidak terbatas
Penghapusan <i>array</i> tidak mungkin	Penghapusan <i>Linked List</i>

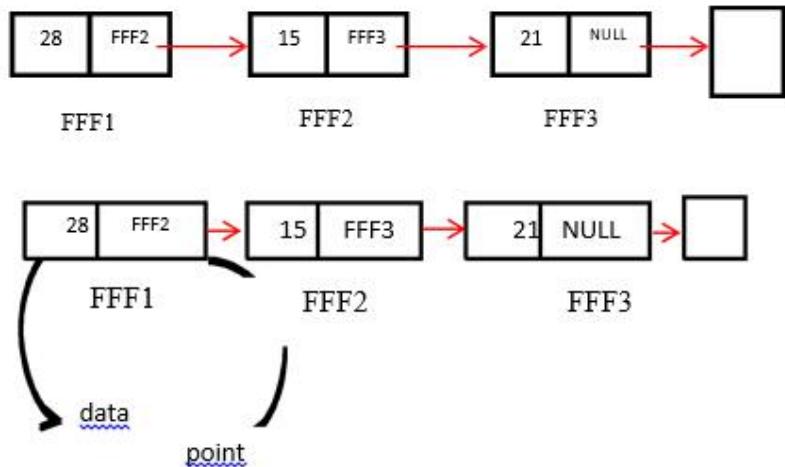
## B. Single Linked List

- 1) Pengertian *Single Linked List*

*Single linked list* merupakan *linked list* yang *field pointer*-nya hanya satu buah saja dan satu arah. Serta pada akhir *node*, *pointernya* menunjuk NULL.



Gambar 6.3 Ilustrasi *Single Linked List*



Gambar 6.4 Ilustrasi *Single Linked List*

Penjelasan :

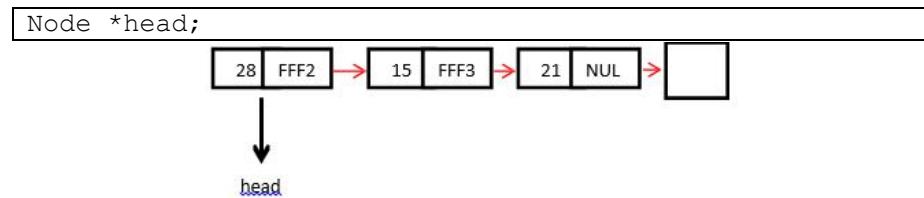
- a) Setiap *node* pada *linked list* mempunyai field yang berisi pointer ke node berikutnya, dan juga memiliki field yang berisi data.
  - b) Node terakhir akan menunjuk ke NULL yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi linked list.
- 2) Jenis *Single List*
- a) *Single linked list* dengan *head*
  - b) *Single linked list* dengan *head* dan *tail*:
- 3) Deklarasi *Single Linked List* (Deklarasi NODE)

Pendeklarasian *Node* pada *single link list* menggunakan *struct* dengan 2 buah *field*, yaitu *field* untuk menyimpan data dan *field* yang bertipe *pointer* dari untuk menyimpan alamat dari *node* selanjutnya. Setelah itu membuat variabel yang bernama *node* yang digunakan kunci untuk mengakses *struct TNode*. Berikut kodennya :

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

4) *Single Linked List Menggunakan Head.*

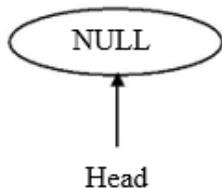
Head akan selalu menunjuk pada *node* pertama. Begini deklarasinya



Gambar 6.5 *Linked List Menggunakan Head*

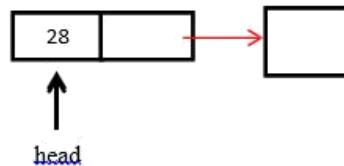
5) Penambahan data dari depan.

- a) *List* masih kosong (*head == NULL*)



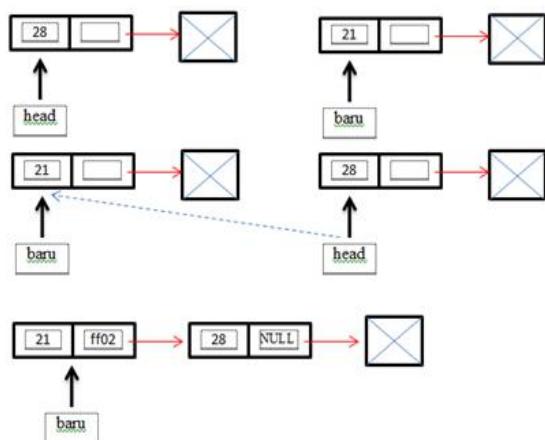
Gambar 6.6 *Head Null*

- b) Masukkan data baru, misal 28



Gambar 6.7 Memasukkan 28

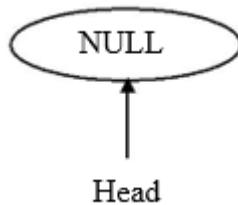
- c) Masukkan data lagi dari depan, misal 21



Gambar 6.8 Memasukkan 21

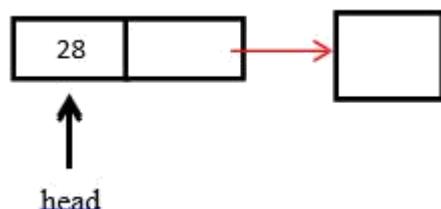
6) Penambahan data dari belakang.

a) List masih kosong ( $head == \text{NULL}$ )



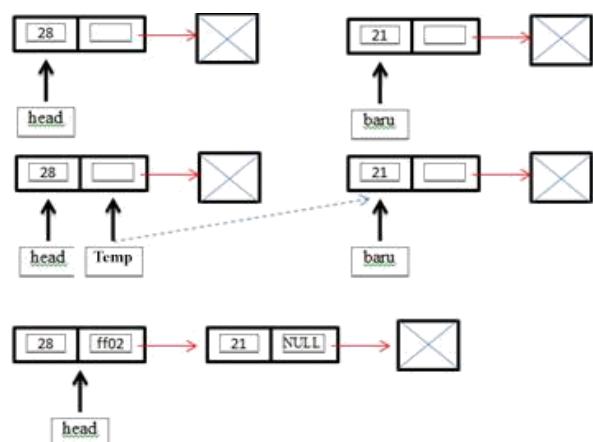
Gambar 6.9 Head Null

b) Masukkan data baru, misal 28



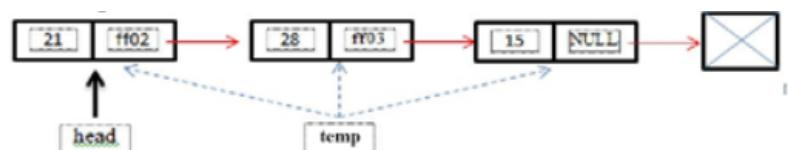
Gambar 6.10 Memasukkan 28

c) Masukkan data lagi, misal 21



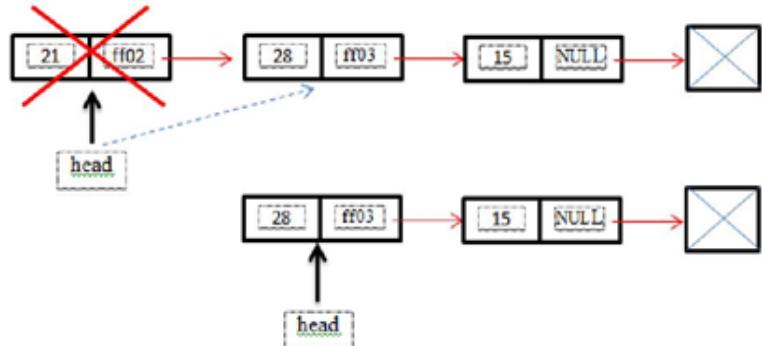
Gambar 6.11 Memasukkan 21

7) Menampilkan isi *linked list*



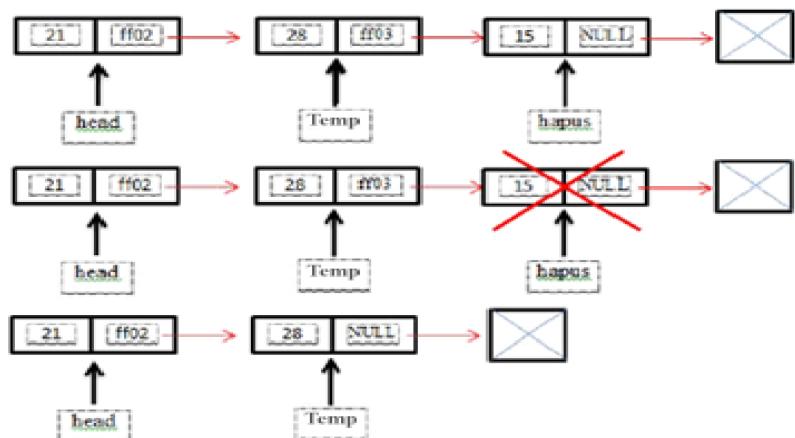
Gambar 6.12 Menampilkan isi *linked list*

8) Menghapus data dari depan



Gambar 6.13 Menghapus data dari depan

9) Menghapus data dari belakang

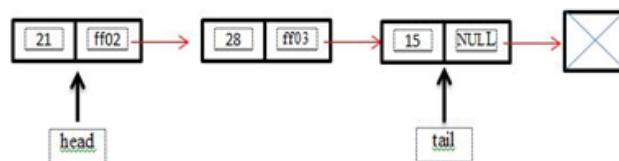


Gambar 6.14 menghapus data dari belakang

10) *Single Linked List* Menggunakan *Head* dan *Tail*

Dalam *single linked list* menggunakan *head* dan *tail* dibutuhkan dua buah *node* bervariabel *pointer* yaitu *head* dan *tail*. *Head* akan selalu menunjuk pada *node* pertama( paling depan) dan *tail* menunjuk pada *node* terakhir (paling belakang ).

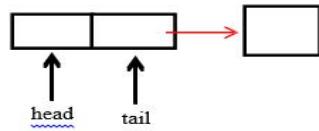
```
Node *head, *tail;
```



Gambar 6.15 *Linked List* Menggunakan *Head* dan *tail*

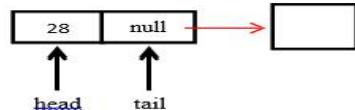
11) Penambahan data dari depan.

- a) List masih kosong ( $head = tail = \text{NULL}$ )



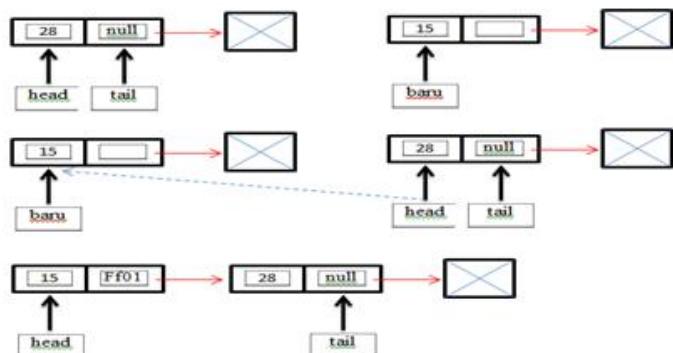
Gambar 6.16 Head & tail Null

- b) Masukkan data baru, misal 28



Gambar 6.17 Memasukkan 28

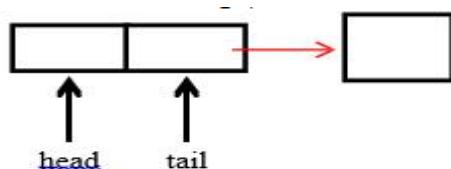
- c) Masukkan data lagi dari depan, misal 15



Gambar 6.18 Memasukkan 15

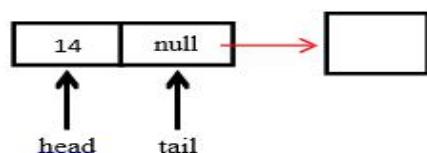
12) Penambahan data dari belakang.

- a) List kosong ( $head = tail = \text{NULL}$ )



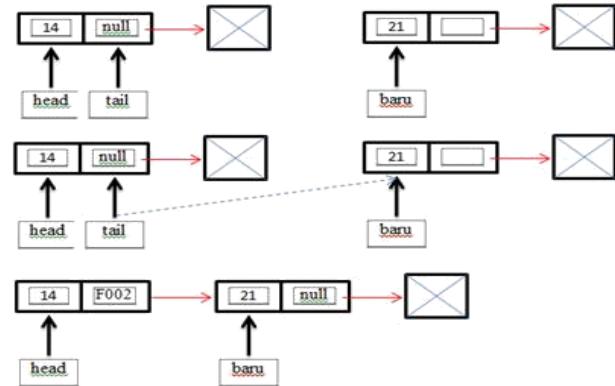
Gambar 6.19 Head & tail Null

- b) Masukkan data baru, misal 14



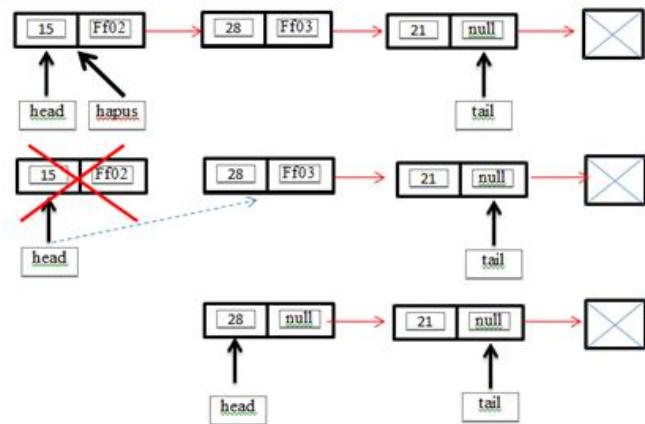
Gambar 6.20 Memasukkan 14

c) Masukkan data lagi, misal 21



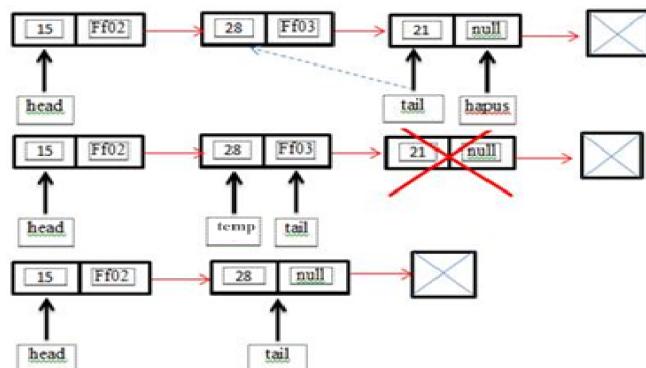
Gambar 6.21 Memasukkan 21

13) Menghapus data dari depan



Gambar 6.22 Menghapus dari depan

14) Menghapus data dari belakang



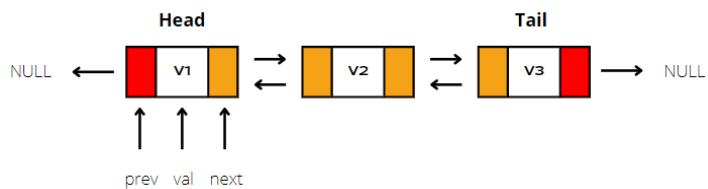
Gambar 6.23 Menghapus data dari belakang

## C. Double Linked List

### 1) Pengertian *Double linked list*

*Double linked list* adalah *linked list* yang *field* pointernya 2 arah.

Merujuk *node* sebelum dan sesudahnya. *Double linked list* memiliki 3 buah *field* dalam 1 node yaitu 1 untuk menyimpan data dan 2 untuk *pointer* (*Next* dan *Prev*). *Pointer Next* menunjuk pada node setelahnya dan *Prev* menunjuk pada *node* sebelumnya.



Gambar 6.24 Ilustrasi *Double Linked List*

Penjelasan :

- Double linked list* merupakan suatu *linked list* yang memiliki dua variabel pointer. Dimana pointer tersebut menunjuk ke *node* sebelum dan selanjutnya (*prev&next*).
- Double linked list* terdiri dari sejumlah elemen (*node*) dimana setiap *node* memiliki penunjuk *prev* (menunjuk *node* sebelumnya) dan *next* (menunjuk *node* selanjutnya).
- Penunjuk *prev* pada *node head* menunjuk ke *NULL*, menandakan bahwa *node head* (*node awal*).
- Penunjuk *next* pada *node tail* menunjuk ke *NULL*, menandakan bahwa *node tail* (*node akhir*).

### 2) Deklarasi & inisialisasi

Untuk membuat sebuah *double linked list* dibutuhkan sebuah *struct*. Berikut adalah deklarasinya:

```
struct LinkedListName{  
    //komponen  
    dataTypeData1 dataName1;  
  
    LinkListName *prev;  
    LinkListName *next;  
};
```

Kemudian untuk inisialisasi pada setiap *node* kurang lebih akan menerapkan seperti berikut:

```
LinkListName *head, *tail;
head = (LinkListName*) malloc(sizeof(LinkListName));
tail = new LinkListName();

head->dataName1 = valData1;
head->prev = NULL;
head->next = tail;

tail->dataName1 = valData1;
tail->prev = head;
tail->next = NULL;
```

### 3) Double Linked List Sederhana

Sebelum kita membuat *double linked list* dengan fungsi sebaiknya kita harus memahami bagaimana dasar dari pembuatan *double linked list* secara sederhana. Silahkan ikuti panduan berikut ini.

#### a) Deklarasi *Double Linked List*

Dalam deklarasi *double linked* pada praktik ini kita akan menggunakan skenario *input* dan *output* data mahasiswa. Silahkan deklarasikan *double linked list* berikut pada awal kode.

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

// Deklarasi double linked list
struct DataMhs{
    string nama, nim, kelas;
    DataMhs *prev;
    DataMhs *next;
};
```

#### b) Inisialisasi *Double Linked List*

Lanjut pada bagian awal fungsi main(), di bagian *inisialisasi* ini tidak terlalu berbeda dengan saat *menginisialisasi single linked list*. Kita akan *inisialisasikan* sebuah variabel *pointer* yang bertipe data *struct* yang telah dibuat.

```
//ingat kode ini ada di dalam main()
//Disini kita buat 2 buah node
DataMhs *node1, *node2;
node1 = new DataMhs();
node2 = new DataMhs();
```

c) Memberi Nilai Variabel Pada *Node1*

Setelah variabel *pointer* telah siap maka selanjutnya kita dapat mengisikan nilai pada variabel tersebut. Ingat, karena *node1* adalah *node* awal atau *node* pertama yang ada, maka *node* ini akan berperan sebagai *head* dan menerapkan penunjuk *prev* pada *node* *head* menunjuk ke NULL, menandakan bahwa *node head* (*node* awal).

```
//letakkan kode ini dibawah inisialisasi
//disini kita akan masukkan data mahasiswa
node1->nama = "Suahartina Sapose";
node1->nim = "981023832";
node1->kelas = "A";
node1->prev = NULL;
node1->next = node2;
```

d) Memberi Nilai Variabel Pada *Node2*

*Node2* berperan sebagai tail karena disini kita hanya membuat 2 buah node. *Node2* menerapkan penunjuk *next* pada node *tail* menunjuk ke NULL, menandakan bahwa node tail (node akhir).

```
//letakkan kode ini dibawah node1
node2->nama = "Kodina Fuzi";
node2->nim = "981572386";
node2->kelas = "B";
node2->prev = node1;
node2->next = NULL;
```

e) Mencetak Double Linked List

Dalam pencetakan *double linked list* kita membutuhkan sebuah variabel *pointer* pembantu yang kita beri nama *currently* disingkat *cur*. Tugas variabel ini akan menyimpan *node* awal kemudian jika setelah dicetak maka akan menyimpan *node* selanjutnya selama variabel *next* yang di simpan pada variabel *cur* tidak *null*. Berikut adalah deklarasi nya.

```
LinkListName *cur;
cur = head;
while(cur != NULL) {
    //Print cout
    cur = cur->next;
}
```

maka dalam kasus yang digunakan kodennya akan seperti berikut :

```
//Letakkan kode ini dibawah node2
//variabel pembantu yang dibuat adalah currently
//dengan tipe data DataMhs
DataMhs *currently;
currently = node1;
while(currently != NULL) {
    cout<<"Nama : "<<currently -> nama << endl;
    cout<<"Nim : "<<currently -> nim << endl;
    cout<<"Kelas: "<<currently -> kelas <<
endl<<endl;
    currently = currently -> next;
}
```

#### 4) Double Linked List Dengan Fungsi

Setelah kita memahami dasar dari implementasi *double linked list*. Selanjutnya kita akan menerapkan penerapan fungsi agar kita tidak perlu menuliskan baris kode secara manual saat ingin melakukan operasi pada program.

##### a) Deklarasi *Double Linked List*

Masih sama, dalam deklarasi *double linked* dengan fungsi pada praktik ini kita akan menggunakan skenario *input* dan *output* data mahasiswa. Silahkan deklarasikan *double linked list* berikut pada awal kode. Terdapat hal yang berbeda yaitu kita juga mendeklarasikan variabel *pointer* lainnya yang dibutuhkan pada *double linked list* yaitu *head*, *tail*, *currently*, *newnode* dan *del*.

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

// Deklarasi Double Linked List
struct DataMhs{
    string nama, nim, kelas;
    DataMhs *prev;
    DataMhs *next;
};

DataMhs *head, *tail, *currently, *newnode, *del;
```

##### b) Fungsi Membuat *Double Linked List*

Untuk membuat *node* baru pada *double link list* kita dapat membuat fungsi *create* dengan parameter data. Disini parameter data merupakan sebuah *array* dengan ukuran index 3 dan tipe datanya sesuai dengan yang digunakan pada *struct* yaitu *string*.

Kemudian variabel *pointer prev* dan *next* tidak menunjuk kepada *node* manapun maka berilah nilai *Null*. Kemudian karena kita membuat sebuah *node* baru pertama kalinya, maka node ini adalah node pertama dan terakhir atau berperan sebagai *node head* dan *node tail*. Deklarasikan *tail = head*.

```
// Fungsi Membuat Node awal
void createdoublelinked(string data[3]){
    head = new DataMhs();
    head -> nama = data[0];
    head -> nim = data[1];
    head -> kelas = data[2];
    head -> prev = NULL;
    head -> next = NULL;
    tail = head;
}
```

### c) Fungsi Mencetak *Double Linked List*

Dalam membuat fungsi print kita juga memberikan sebuah *exception*. *Exception* ini akan berguna jika kita ingin mencetak sebuah *double link list* namun tidak ada *double link list* yang tersedia. Kemudian setelah *exception* barulah kita deklarasikan perulangan *while* yang akan mencetak nilai *double link list*. Kita dapat menggunakan fungsi cetak pada penerapan *double link list* tanpa fungsi.

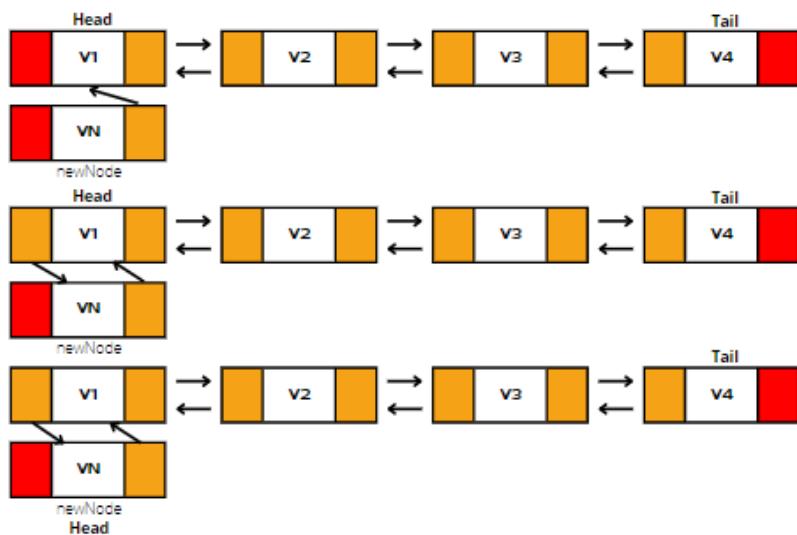
```
//Fungsi mencetak double linked list
void printdoublelinked(){
    if (head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        cout<<"Isi Data : "<<endl;
        currently = head;
        while(currently != NULL){
            cout<<"Nama : "<<currently -> nama
            << endl;
            cout<<"Nim : "<<currently -> nim
            << endl;
            cout<<"Kelas : "<<currently ->
            kelas << endl << endl;
            currently = currently -> next;

        }
    }
}
```

Dengan cukup menyediakan 3 buah fungsi ini kita dapat mengisikan nilai pada sebuah *node double link list* dan mencetaknya. Panggil fungsi yang ada pada fungsi ini di *main()*.

```
int main(){
    //membuat node baru
    string newdata[3] = {"Suahartina Sapose",
    "981023832", "A"};
    createdoublelinked(newdata);
    printdoublelinked();
}
```

d) Fungsi Menambahkan *Node* Baru Pada Awal *Node*



Gambar 6.25 Ilustrasi tambah *node* depan

Penjelasan pada gambar di atas yaitu, bilamana kita mempunyai sekumpulan *node* dan kita ingin menambahkan *node* baru didepannya. Maka kita asumsikan pada *node* terbaru variabel *pointer prev* nya menunjuk ke *Null* dan variabel *pointer next* nya menunjuk ke *node head*. Setelah itu, pada *node head* kita ubah nilai variabel *pointer prev* nya dengan *node* baru. Terakhir pada *node* terbaru kita atur menjadi *head*.

```
//Fungsi tambah dari depan
void addfirst(string data[4]){
    if (head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        newnode = new DataMhs();
        newnode -> nama = data[0];
        newnode -> nim = data[1];
        newnode -> kelas = data[2];
        newnode -> prev = NULL;
        newnode -> next = head;
    }
}
```

```

        head->prev = newnode;
        head = newnode;
    }
}

```

Selanjutnya fungsi dapat digunakan pada fungsi *main()*. Silahkan tambahkan sebuah variabel data baru kemudian gunakan variabel data tersebut menjadi parameter pada fungsi ini di *main()*.

```

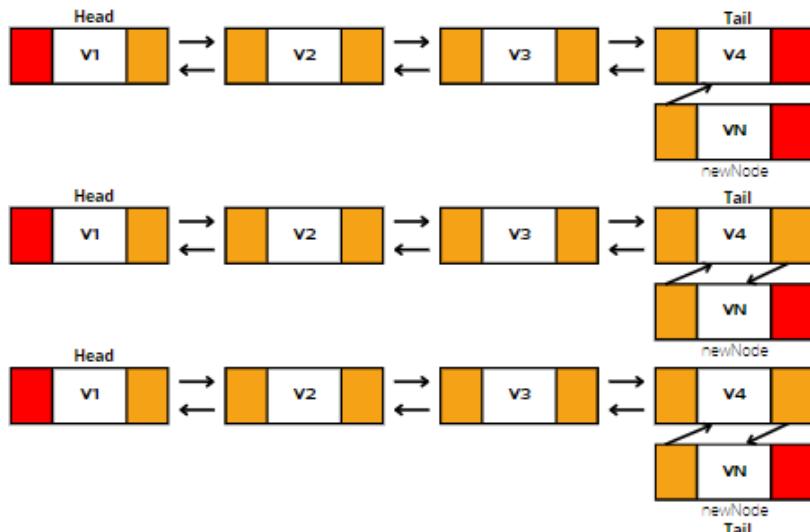
int main(){
    //membuat node baru
    string newdata[3] = {"Suahartina Sapose",
    "981023832", "A"};
    createdoublelinked(newdata);
    printdoublelinked();

    //penambahan node dari depan
    string data2[3] = {"Kodina Fuzi", "981572386",
    "B"};
    addfirst(data2);
    printdoublelinked();
}

```

#### e) Fungsi Menambahkan *Node* Baru Pada Akhir *Node*

Untuk dapat menambahkan data pada akhir *node* konsep nya hampir sama dengan menambahkan data pada awal *node*.



Gambar 6.26 Ilustrasi tambah node belakang

Penjelasan pada gambar diatas yaitu, bilamana kita mempunyai sekumpulan *node* dan kita ingin menambahkan *node* baru setelah *node* terakhir. Maka kita asumsikan pada *node* terbaru variabel *pointer prev* nya menunjuk ke *Tail* dan variabel *pointer next* nya menunjuk ke *Null*. Setelah itu, pada *node tail* kita ubah nilai

variabel *pointer Next* nya dengan *node* baru. Terakhir pada *node* terbaru kita atur menjadi *tail*.

```
//Fungsi tambah dari belakang
void addlast(string data[4]){
    if (head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        newnode = new DataMhs();
        newnode -> nama = data[0];
        newnode -> nim = data[1];
        newnode -> kelas = data[2];
        newnode -> prev = tail;
        newnode -> next = NULL;
        tail->next = newnode;
        tail = newnode;
    }
}
```

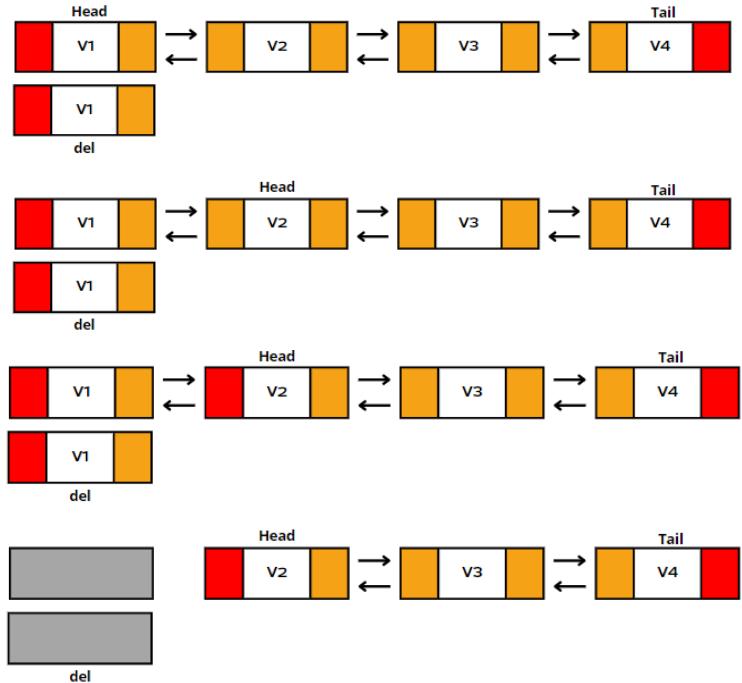
Selanjutnya fungsi dapat digunakan pada fungsi *main()*. Silahkan tambahkan sebuah variabel data baru kemudian gunakan variabel data tersebut menjadi parameter pada fungsi ini di *main()*.

```
int main(){
    string newdata[3] = {"Suahartina Sapose",
"981023832", "A"};
    createdoublelinked(newdata);
    printdoublelinked();

    string data2[3] = {"Kodina Fuzi", "981572386",
"B"};
    addfirst(data2);
    printdoublelinked();

    string data3[4] = {"Daytalulah Suksis",
"981143946", "C"};
    addlast(data3);
    printdoublelinked();
}
```

f) Fungsi Menghapus *Node Depan*

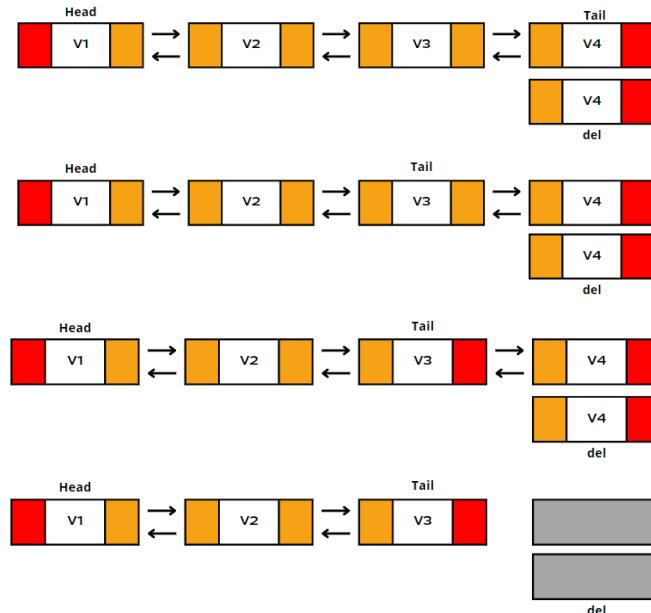


Gambar 6.27 Ilustrasi hapus *node head*

Dalam melakukan penghapusan *node*, kita membutuhkan sebuah *node* pembantu yaitu *delete* atau disingkat *del*. Variabel *pointer* *del* merupakan duplikasi dari *head*. Kemudian, kita atur *head* dipindahkan ke *node* ke-2 (*node* yang ditunjuk *Next* pada *node* yang ingin dihapus). Kemudian pada *head* terbaru kita ubah *prev* nya menjadi *Null*. Terakhir, kita sudah aman untuk menghapus variabel *del* dan *node* yang ingin dihapus (*head* lama).

```
//Fungsi hapus dari depan
void removefirst(){
    if(head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        del = head;
        head = head->next;
        head->prev = NULL;
        delete del;
    }
}
```

g) Fungsi Menghapus *Node* Belakang



Gambar 6.28 Ilustrasi hapus *node tail*

Tidak jauh berbeda, penghapusan *node* dari belakang konsepnya mirip dengan penghapusan dari depan. Kita membutuhkan sebuah *node* pembantu yaitu *delete* atau disingkat *del*. Variabel *pointer* *del* merupakan duplikasi dari *tail*. Kemudian, kita atur *tail* dipindahkan ke *node* sebelum *tail* (*node* yang ditunjuk *Prev* pada *node* yang ingin dihapus). Kemudian pada *tail* terbaru kita ubah *next* nya menjadi *Null*. Terakhir, kita sudah aman untuk menghapus variabel *del* dan *node* yang ingin dihapus (*tail* lama).

```
//Fungsi hapus dari belakang
void removelast(){
    if(head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        del = tail;
        tail = tail->prev;
        tail->next = NULL;
        delete del;
    }
}
```

## 6.5 Tugas Praktikum ke-1: Program singgel linked list

*Source Code :*

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

struct tnode{
    int data;
    tnode *next;
};

tnode *head;
void init(){
    head=NULL;
}

int iskosong(){
    if(head == NULL){
        return 1;
    }
    else{
        return 0;
    }
}

void tampil(){
    tnode *bantu;
    bantu=head;
    if(iskosong()==0)
    {
        while(bantu!=NULL)
        {
            cout<<bantu->data<<" ";
            bantu=bantu->next;
        }
        cout<<"\n";
    }
    else
    {
        cout<<"masih kosong\n";
    }
}

void isidepan(int databaru){
    tnode *baru;
    baru = new tnode;
    baru->data = databaru;
    baru->next = NULL;
    if(iskosong()==1){
        head=baru;
        head->next=NULL;
    }
    else{
        baru->next=head;
        head=baru;
    }
    cout<<"Penambahan data berhasil\n";
}
```

```

void isibelakang(int databaru){
    tnode *baru,*bantul;
    baru = new tnode;
    baru->data = databaru;
    baru->next = NULL;
    if(iskosong()==1) {
        head=baru;
        head->next=NULL;
    }else{
        bantul = head;
        while(bantul->next!=NULL) {
            bantul = bantul->next;
        }
        bantul->next = baru;
    }
    cout<<"penambahan data berasil\n";
}

void hapus_depan(){
    tnode *hapus;          // pointer bertipe tnode dengan nama =
'hapus'
    int d;                // mendefinisikan variabel d bertipe
integer.
    if(iskosong()==0)    // if ( 0 = 0 )
    {
        if(head->next!=NULL)      // (FF03->next!=null) -- (FF01
!= null) -- ya
        {
            hapus=head;           // hapus = FF03
            d=hapus->data;         // d = FF03->data -- d=8
            head=head->next;       // head = FF03->next -- head =
FF01
            delete hapus;          // delete FF03
        }
        else
        {
            d=head->data;
            head=NULL;
        }
        cout<<d<<" "<<"terhapus\n";
    }
    else
    {
        cout<<"masih kosong";
    }
}

void hapus_belakang()
{
    tnode *hapus,*bantu;
    int d;

    if(iskosong()==0)
    {
        if(head->next!=NULL)
        {
            bantu = head;
            while(bantu->next->next!=NULL)

```

```

        {
            bantu = bantu->next;
        }
        hapus = bantu->next;
        d=hapus->data;
        bantu->next=NULL;
        delete hapus;
    }

    else
    {
        d=head->data;
        head=NULL;
    }
    cout<<d<<" "<<"terhapus\n";
}
else
{
    cout<<"masih kosong";
}
}

int main()
{
    int pilihan, data;
    do
    {
        cout<<endl;
        cout<<"2118131 | Moh harifin"<<endl;
        cout<<"======"<<endl;
        cout<<"Single Linked List dengan Head"<<endl;
        cout<<"1. Masukkan Data dari Depan"<<endl;
        cout<<"2. Masukkan Data dari Belakang"<<endl;
        cout<<"3. Hapus Data dari depan"<<endl;
        cout<<"4. Hapus Data dari belakang"<<endl;
        cout<<"5. Tampil Data"<<endl;
        cout<<"6. Keluar"<<endl;
        cout<<"Masukkan Pilihan Anda : ";
        cin>>pilihan;
        if (pilihan == 1 )
        {
            cout<<"Masukkan Data = "; cin>>data;
            isidepan(data);
        }
        else if (pilihan == 2 )
        {
            cout<<"Masukkan Data = "; cin>>data;
            isibelakang(data);
        }
        else if (pilihan == 3 )
        {
            hapus_depan();
        }
        else if (pilihan == 4 )
        {
            hapus_belakang();
        }
        else if (pilihan == 5 )
        {
    }
}

```

```

        tampil();
    }
    else if (pilihan == 6 )
    {
        cout<<"Keluar dari program";
    }
    else
    {
        cout<<"Pilihan tidak tersedia... \n\n";
    }
}
while(pilihan !=6 );
}

```

Tampilan program :

```

tikum $ Data/per8/" && g++ linked_list.cpp -o linked_list &
/DATA KULIAH/semester 2/Praktikum S.Data/per8/"linked_list

2118131 | Moh harifin
=====
Single Linked List dengan Head
1. Masukkan Data dari Depan
2. Masukkan Data dari Belakang
3. Hapus Data dari depan
4. Hapus Data dari belakang
5. Tampil Data
6. Keluar
Masukkan Pilihan Anda : 1
Masukkan Data = 2118132
Penambahan data berhasil

2118131 | Moh harifin
=====
Single Linked List dengan Head
1. Masukkan Data dari Depan
2. Masukkan Data dari Belakang
3. Hapus Data dari depan
4. Hapus Data dari belakang
5. Tampil Data
6. Keluar
Masukkan Pilihan Anda : 1
Masukkan Data = 2118130
Penambahan data berhasil

2118131 | Moh harifin
=====
Single Linked List dengan Head
1. Masukkan Data dari Depan
2. Masukkan Data dari Belakang
3. Hapus Data dari depan
4. Hapus Data dari belakang
5. Tampil Data
6. Keluar
Masukkan Pilihan Anda : 2
Masukkan Data = 2118131
penambahan data berhasil

```

Gambar 6.29 Tampilan Program *singgel linked list*

Analisa Program :

Pada analisa program di atas merupakan program singgel *linked list*. Saat program dijalankan yaitu akan menampilkan sebuah pilihan yang di mana pilihan tersebut yang pertama menambah data dari depan dan yang kedua menambah data dari belakang dan ketiga menghapus data dari depan dan yang keempat menghapus data dari belakang dan ke lima menampilkan data dan yang keenam yaitu keluar. Selanjutnya yaitu user akan menginputkan pilihan mana yang akan di lakukan pemrosesan. Jika *user* memilih satu maka akan menjalankan *if (pilihan==1)* yaitu menambahkan data dari depan dan langsung akan memasukkan data dan akan melakukan pemanggilan *function* isi data

dengan parameter data. Jika terjadi pemanggilan maka akan menjalankan program pada baris yang ke 40 hingga 54. Di dalam *function* pada isi depan terdapat *statement* *tinode* \*baru yang akan melakukan penyimpanan alamat dari data baru dan pada *tmode* juga pada program baris 4 keempat untuk mendefinisikan *struct tnode* yang akan melakukan penyimpanan dari *pointer* data *next (tnod \*next)*. Selanjutnya yaitu kembali pada baris 42 yang di mana baru = *new tnode* merupakan data baru dan akan berlanjut baru->data = databaru dan ->next = null. Selanjutnya yang akan melakukan pengecekan apakah *if (iskosong()==1)* jika ya maka akan menjalankan *statament* yang ada di dalamnya yaitu baru->next=head dan head=baru, jika tidak maka akan menjalankan *else* yang di dalamnya terdapat *statement* baru->next=head dan head=baru. Di iskosong tersebut pada baris yang 13 di dalamnya terdapat statement apakah *if(head==null)* jika ya return1 *jikan* tidak *return* 0. Dan selanjutnya akan menampilkan penambahan data berhasil. Selanjutnya jika user memilih dua yaitu memasukkan data dari belakang maka akan menjalankan *if (pilihan==2)* dengan *statement* di dalamnya memasukkan data dan akan melakukan pemanggilan *function* isibelakang dengan parameter data. Pada *fancction* isibelakang si baris 56 yang di dalamnya melakukan pemanggilan tipe data dari *struct* *tnode* yaitu *tnode* dengan *pointer* baru dan *pointer* bantu1. Dan baru= *new tnode* untuk melakukan penambahan dan baru->data= databaru, baru->next=null. Dan akan melakukan pengecekan apakah *if (iskosong()==1)* jika iya maka head=baru dan head->next=null, jika tidak maka bantu1=head dan akan melakukan perulangan *while ( bantu->next!=null)* jika ya maka bantu1=bantu1->next dan akan berlanjut->next=baru dan penambahan dat berhasil. Selanjutnya yaitu jika user memilih tiga yaitu hapus data depan maka akan menjalankan *if (pilihan==3)* dengan melakukan pemanggilan *hapus\_depan()*. Didalam *function* *hapus\_depan* pada baris yang ke 74 yang dimana terdapat pemanggilan *tnode* dengan *pointer* *hapus* dan akan melakukan pengecekan apakah *if(iskosong()==0)* jika ya maka akan melakukan pengecekan lagi yaitu *if (head->next!=null)* jika ya maka akan menjalankan *statement* yang ada di dalamnya jika tidak maka akan menjalankan *statement* yang ada di dalamnya dan akan menampilkan terhapus tapi jika if

(iskosong()==0) tidak maka akan menampilkan data masih kosong. Selanjutnya yaitu jika *user* memilih 4 yaitu untuk menghapus data depan maka akan menjalankan *if(pilihan==4)* dengan *statement* di dalamnya melakukan pemanggilan hapus belakang(). Pada *function* hapus belakang pada baris 100 yaitu terdapat tipe data *tnode* dengan *pointer* hapus dan pointer bantu. Selanjutnya akan melakukan pengecekan *if(iskosong()==0)* jika iya maka akan menjalankan *statement* yang di dalamnya dan akan melakukan pengecekan lagi yaitu *if(head->next!=null)* jika ya maka akan menjalankan *statement* yang di dalamnya. Di dalam *statement* tersebut terdapat pendeklarasian *bantu =head* dan akan berlanjut ke perulangan *while (bantu->next->next!=null)* jika ya maka *bantu =bantu->next* jika tidak maka program akan berlanjut ke program selanjutnya yaitu *hapus = bantu->next* dan *d=bantu->next* dan *bantu->next=null* dan *delete* hapus maka akan menampilkan terhapus jika tidak maka akan menjalankan *statement* yang ada di dalamnya *else* yaitu *d=head->data* dan *head =null*. Jika apakah kondisi *if(iskosong()==0)* tidak maka akan menjalankan *else* dan akan menampilkan data masih kosong. Selanjutnya jika *user* memilih lima yaitu tampil data maka akan melakukan pemanggilan tampil(). Pada *function* tampil tersebut akan melakukan pemanggilan dari tipe data *struct tnode* dengan *pointer \*bantu* dan *bantu=head* dan akan melakukan pengecekan *if(iskosong()==0)* jika tidak maka akan menampilkan masih kosong tapi jika iya maka akan menjalankan *statement* yang ada di dalamnya. Yang ada dalam *statement* yang ada di dalamnya terdapat perulangan *while (bantu!=null)* dengan *statement cout<<bantu->data<<" "* dan *bantu=bantu->next*. Selanjutnya jika *user* memilih 6 yaitu keluar maka program berakhir dan tidak akan melakukan perulangan program selesai.

## 6.6 Tugas Praktikum ke-2: Doble linked list data pondok

*Source Code :*

```

# include <iostream>
using namespace std;
// Deklarasi Double Linked List
struct DataMhs{
    string nama,kode,jenis,angkatan;

    DataMhs *prev;
    DataMhs *next;
};

float prev=0 ,next=0;
void garis(){
    cout<<"_____"<<endl;
}

DataMhs *head, *tail, *currently, *newnode, *del;
// Fungsi Membuat Node awal
void createdoublelinked(string data[4]){
    head = new DataMhs();
    head -> nama = data[0];
    head -> kode = data[1];
    head -> jenis = data[2];
    head -> angkatan = data[3];
    head -> prev = NULL;
    head -> next = NULL;
    tail = head;
}

//Fungsi mencetak double linked list
void printdoublelinked(){
    if (head == NULL){

    }else{
        cout<<"Cetak Data : "<<endl;
        currently = head;
        while(currently != NULL){
            cout<<endl<<endl;
            cout<<"\t\tAlamat Node : "<<currently<<endl;

        cout<<"_____"<<endl;
        cout<<"| "<<currently->prev<<"| ";
        cout<<"\t Nama : "<<currently -> nama ;
        cout<<"\t\t| "<<currently->next<<"| "<<endl;
        cout<<"\t\t| Kode kamar : "<<currently ->
kode <<"\t| "<<endl;
        cout<<"\t\t| jenis kelamin : "<<currently ->
jenis <<"\t| "<<endl;
        cout<<"\t\t| angkatan : "<<currently ->
angkatan<<"\t| "<<endl ;

        cout<<"_____"<<endl;
        garis();
        currently = currently -> next;
    }
}

```

```

    }

//Fungsi tambah dari depan
void addfirst(string data[4]){
    if (head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        newnode = new DataMhs();
        newnode -> nama = data[0];
        newnode -> kode = data[1];
        newnode -> jenis = data[2];
        newnode -> angkatan = data[3];
        newnode -> prev = NULL;
        newnode -> next = head;
        head->prev = newnode;
        head = newnode;
    }
}

//Fungsi tambah dari belakang
void addlast(string data[4]){
    if (head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        newnode = new DataMhs();
        newnode -> nama = data[0];
        newnode -> kode = data[1];
        newnode -> jenis = data[2];
        newnode -> angkatan = data[3];
        newnode -> prev = tail;
        newnode -> next = NULL;
        tail->next = newnode;
        tail = newnode;
    }
}

//Fungsi hapus dari depan
void removefirst(){
    if(head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        del = head;
        head = head->next;
        head->prev = NULL;
        delete del;
    }
}

//Fungsi hapus dari belakang
void removelast(){
    if(head == NULL){
        cout<<"Double linked list belum ada";
    }else{
        del = tail;
        tail = tail->prev;
        tail->next = NULL;
        delete del;
    }
}

```

```

}

int main (){
    int pilihan, pilihan2;
    string newdata [4];
    string nama,kode,jenis,angkatan;
    do
    {
        cout<<endl;
        cout<<"| 2118131 |      Mohammad harifin
|<<endl;
        garis();
        cout<<"| Selamat Datang di Pondok Pesantren
|<<endl;
        cout<<"| 1. Buat double linked list
|<<endl;
        cout<<"| 2. Keluar
|<<endl;
        garis();
        cout<<"Masukkan pilihan : ";
        cin>>pilihan;
        cout<<endl<<endl;
        if (pilihan == 1 )
        {
            cout<<"Masukkan Nama : ";
            cin>>nama;
            newdata[0]=nama;
            cout<<"Masukkan Kode kamar (G.L.R) : ";
            cin>>kode;
            newdata[1]=kode;
            cout<<"Jenis Kelamin (L/P) : ";
            cin>>jenis;
            newdata[2]=jenis;
            cout<<"Angkatan : ";
            cin>>angkatan;
            newdata[3]=angkatan;
            createdoublelinked(newdata);
            cout<<"Selamat...Anda telah membuat double linked
list "<<endl;
            do
            {
                cout<<endl;
                cout<<"| 2118131 |      Mohammad harifin
|<<endl;
                garis();
                cout<<"| Double lingked list
|<<endl;
                cout<<"| 1. AddFirst
|<<endl;
                cout<<"| 2. AddLast
|<<endl;
                cout<<"| 3. RemoveFirst
|<<endl;
                cout<<"| 4. RemoveLast
|<<endl;
                cout<<"| 5. Cetak
|<<endl;
                cout<<"| 6. Keluar
|<<endl;

```

```

        garis();
        cout<<"Masukkan Pilihan anda : ";
        cin>pilihan2;
        if (pilihan2==1)
        {
            cout<<"Tambahkan Node dari depan "
            <<endl;
            garis();
            cout<<"Masukkan Nama : ";
            cin>nama;
            newdata[0]=nama;
            cout<<"Masukkan Kode kamar (G.L.R) : ";
            cin>>kode;
            newdata[1]=kode;
            cout<<"Jenis Kelamin (L/P) : ";
            cin>>jenis;
            newdata[2]=jenis;
            cout<<"Angkatan : ";
            cin>>angkatan;
            newdata[3]=angkatan;
            addfirst (newdata);
        }else if (pilihan2 == 2)
        {
            cout<<"Tambahkan Node dari belakang
"=<<endl ;
            garis();
            cout<<"Masukkan Nama : ";
            cin>nama;
            newdata[0]=nama;
            cout<<"Masukkan Kode kamar (G.L.R) : ";
            cin>>kode;
            newdata[1]=kode;
            cout<<"Jenis Kelamin (L/P) : ";
            cin>>jenis;
            newdata[2]=jenis;
            cout<<"Angkatan : ";
            cin>>angkatan;
            newdata[3]=angkatan;
            addlast (newdata);
        }
        else if (pilihan2 == 3)
        {
            removefirst();
        }
        else if (pilihan2 == 4)
        {
            removelast();
        }
        else if (pilihan2 == 5)
        {
            printdoublelinked();
        }
        else if (pilihan2 == 6)
        {
            cout<<"Keluar dari program";
        }
        else
        {
            cout<<"Pilihan tidak tersedia... \n\n";
        }
    }
}

```

```

        }

    } while (pilihan2!=6);

}

else if (pilihan == 2)
{
    cout<<"Keluar dari program";
}
else
{
    cout<<"Pilihan tidak tersedia... \n\n";
}
} while (pilihan !=2);

}

```

Tampilan program :

Alamat Node : 0x600000264000		
0x0	Nama : Aisyah  0x600000260000	
Kode kamar : II.6.14		
jenis kelamin : Perempuan		
angkatan : 2022		
<hr/>		
Alamat Node : 0x600000260000		
0x600000264000	Nama : Arifin  0x600000268000	
Kode kamar : III.4.12		
jenis kelamin : Laki-laki		
angkatan : 2021		
<hr/>		
Alamat Node : 0x600000268000		
0x600000260000	Nama : Zahra  0x0	
Kode kamar : II.4.15		
jenis kelamin : Perempuan		
angkatan : 2022		

Gambar 6.30 Tampilan Program doble *linked list*

Analisa Program :

Pada analisa program di atas yang merupakan program membuat doble *linked list* dengan tema pondok pesantren. Di dalam program tersebut terdapat variabel yang bertipe data integer pada pilihan dan pilihan2 dan juga ada yang bertipe data *string* yaitu *newdata* [4], nama, kode, jenis dan angkatan. Pada saat program di jalankan maka akan menampilkan dua pilihan, yang pertama yaitu buat *double linked list* dan yang ke dua keluar dengan menggunakan *do wile*. Jika *user* memilih 1 maka akan menuju ke if dengan kondisi if(pilihan==1) dan *user* akan melakukan *inputan* nama, kode kamar, jenis kelamin dan angkatan yang akan di simpan dalam *array new data*. Dan saat user sudah input maka akan di lakukan pemanggilan *function createdoublelinked(newdata)*. Pada

*function createdoublelinked* dengan parameter variabel *array* dengan tipe data *string*. Yang ada dalam statement tersebut yaitu terdapat statement yang dimana *head* = *new DataMhs()* yang di mana *DataMhs* tersebut adalah *struct* pendeklarasian dari *doble linked list*, dan *head -> nama* = *data[0]* untuk menyimpan nama dalam *head* dan begitu juga seterusnya. Jika *user* sudah menginputkan yang sebagai data baru dari *double linked list* maka selanjutnya yaitu menampilkan pilihan dalam *double lingked list* tersebut dengan perulangan *do while* dan *if else* pada pilihan tersebut. Dan dalam pilihan *double lingked list* tersebut yang pertama yaitu *AddFirst* yang kedua *AddLast* dan yang ketiga *RemoveFirst* dan yang keempat *RemoveLast* yang kelima Cetak dan yang terakhir yang keenam yaitu Keluar. Pada saat *user* memilih satu maka akan melakukan *inputan* yang sama seperti sebelumnya hanya saja inputan yang *user* lakukan yaitu menambahkan *node* depan. Pada saat *user* sudah meng *inputkan* maka akan melakukan pemanggilan pada *function addfirst* di baris yang ke 53 dengan parameter variabel data *array* dengan tipe data *string*. Di dalam *statement addfirst* akan melakukan pengecekan dengan menggunakan if. Pada if tersebut terdapat sebuah kondisi yang dimana jika if (*head == NULL*) maka akan menampilkan cout<<"*Double linked list* belum ada". Jika tidak maka *newnode* = *new DataMhs()* dan *newnode -> nama* = *data[0]* dan *newnode -> kode* = *data[1]* dan *newnode -> jenis* = *data[2]* dan *newnode -> angkatan* = *data[3]* dan *newnode -> prev* = *NULL* dan *newnode -> next* = *head* dan *head -> prev* = *newnode* dan *head = newnode*. Jika *user* sudah meng inputkan maka akan kembali ke pilihan sebelumnya. Jika *user* memilih yang ke dua maka akan menginputkan sebelumnya hanya saja *input* tersebut menambahkan *node* dari belakang jadi jika *user* sudah menginputkan maka akan melakukan pemanggilan *function addlast* untuk melakukan penambahan di bagian belakang. Pada *function addlast* di baris ke 70 dengan parameter variabel data *array* yang bertipe data *string* di dalam *statement addlast* tersebut terdapat pengecekan dengan menggunakan *if else* yang di mana jika if (*head == NULL*) maka akan menampilkan cout<<"*Double linked list* belum ada". Jika tidak *newnode* = *new DataMhs()* dan *newnode -> nama* = *data[0]* dan *newnode -> kode* = *data[1]* dan *newnode -> jenis* = *data[2]* dan *newnode -> angkatan* =

data[3] dan newnode -> prev = tail dan newnode -> next = NULL dan tail->next = newnode dan tail = newnode. Jika user sudah menginputkan maka akan kembali ke menu sebelumnya untuk melakukan pemilihan. Jika user memilih tiga maka akan melakukan penghapusan di bagian dengan memanggil *function removefirst* pada kondisi if(pilihan2==3). Di dalam *statement* tersebut yaitu akan melakukan pengecekan dengan menggunakan *if else*. Jika *if(head == NULL)* bernilai *true* maka akan menampilkan *statement* yang didalamnya jika *false* maka akan menjalankan *statement* berikutnya untuk melakukan penghapusan dengan ketentuan yang sudah di tentukan. Jika user memilih empat maka akan melakukan penghapusan di bagian dengan memanggil *function removelast* pada kondisi if(pilihan2==4). Di dalam *statement* tersebut yaitu akan melakukan pengecekan dengan menggunakan *if else*. Jika *if(head == NULL)* bernilai *true* maka akan menampilkan *statement* yang di dalamnya jika *false* maka akan menjalankan *statement* berikutnya untuk melakukan penghapusan dengan *ketentuan* yang sudah di tentukan. Dan jika user memilih 5 maka akan melakukan pemanggilan *function printdoublelinked()* dan di dalamnya terdapat statement untuk melakukan pengecekan dengan menggunakan *if else*. Jika *if(head == NULL)* bernilai *true* maka akan menjalankan *statement* yang ada di dalamnya. Jika *false* maka akan menjalankan *statement* yang ada di dalamnya yang di mana *currently = head* dengan berlanjut ke perulangan *while* dengan kondisi *while(currently != NULL)* menampilkan yang sudah di inputkan. Jika memilih enam maka akan keluar dari program dan program selesai.

## 6.7 Kesimpulan

1. *Linked list* adalah sekumpulan elemen dari data yang bertipe sama yang saling terurut dan terhubung dengan bantuan variabel *pointer*, yang setiap data di dalamnya disebut *node* ( simpul ) menempati alokasi memori secara dinamis dan biasanya berupa *struct* yang terdiri dari beberapa *field*.
2. *Single linked list* merupakan *linked list* yang *field pointer*-nya hanya satu buah saja dan satu arah.
3. *Double linked list* adalah *linked list* yang *field pointer*nya 2 arah. Merujuk *node* sebelum dan sesudahnya.

Disetujui Aslab Alfina (2018041) Tgl:	Ttd / Paraf
---	-------------

## BAB 7

# TREE

### 7.1 Jumlah Pertemuan

2 x 50 menit.

### 7.2 Tujuan

1. Praktikan dapat memahami pengertian dari *Tree*
2. Agar praktikan mengetahui dan memahami cara kerja dari *Tree*
3. Praktikan dapat mengoprasikan atau menerapkan metode *Tree*

### 7.3 Alat dan Bahan

1. Perangkat komputer
2. Perangkat lunak: Dev C++
3. Modul Struktur Data 2022

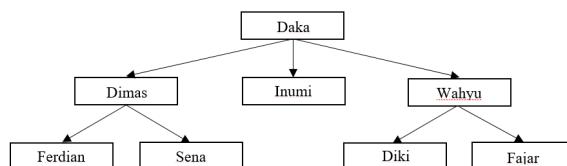
### 7.4 Landasan Teori

#### A. Pengertian Tree

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

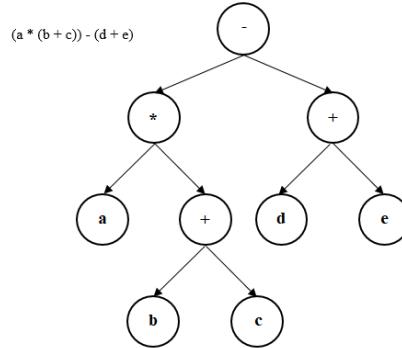
#### B. Contoh Tree

##### 1) Pohon Keluarga



Gambar 7.1 Pohon Keluarga

## 2) Parse Tree

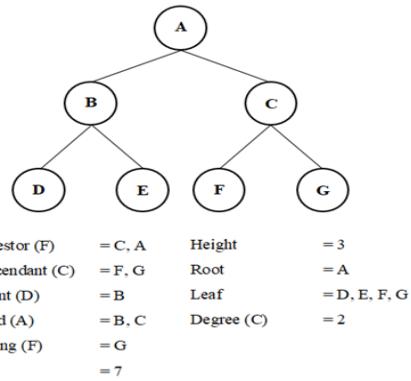


Gambar 7.2 Parse Tree

### C. Istilah dalam Tree

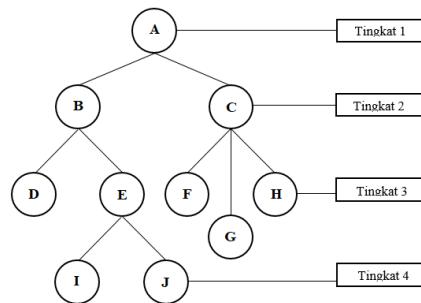
Tabel 7.1 Istilah

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



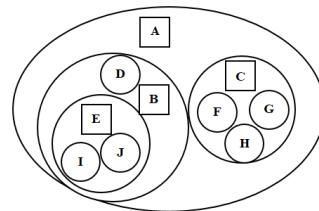
Gambar 7.3 Contoh Istilah Dalam Tree

#### D. Representasi Tree



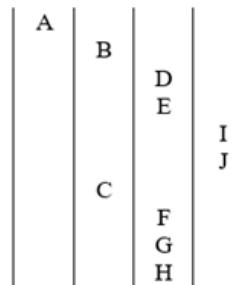
Gambar 7.4 Representasi Tree

##### 1) Diagram Venn



Gambar 7.5 Diagram Venn

##### 2) Notasi Tingkat



Gambar 7.6 Notasi Tingkat

3) Notasi Kurung

$$(A(B(D,E(I,J)),C(F,G,H)))$$

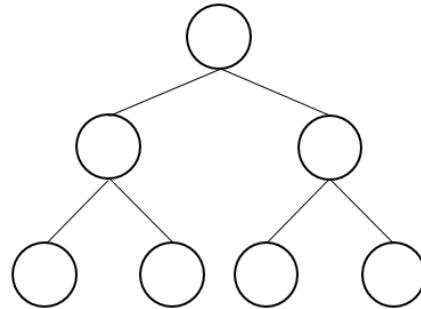
Gambar 7.7 Notasi Kurung

## E. Binary Tree

*Binary Tree* adalah *tree* dengan syarat bahwa tiap *node* hanya boleh memiliki maksimal dua *subtree* dan kedua *subtree* tersebut harus terpisah. Sesuai dengan definisi tersebut tiap node dalam *binary tree* hanya boleh memiliki paling banyak dua *child*. Jenis-jenis *binary tree* :

1) *Full Binary Tree*

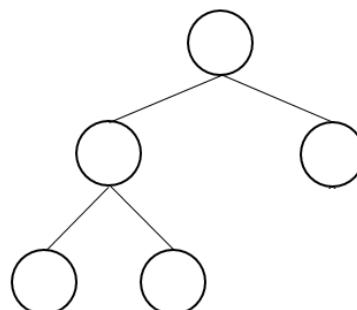
*Binary Tree* yang tiap *node*nya (kecuali *leaf*) memiliki dua *child* dan tiap *subtree* harus mempunyai panjang *path* yang sama.



Gambar 7.8 Full Binary Tree

2) *Complete binary tree*

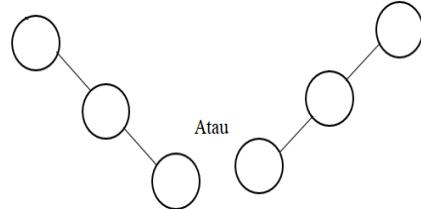
Mirip dengan *Full Binary Tree*, namun tiap *subtree* boleh memiliki panjang *path* yang berbeda. Node kecuali *leaf* memiliki 0 atau 2 *child*.



Gambar 7.9 Complete Binary Tree

### 3) Skewed binary tree

*Binary Tree* yang semua *node*nya (kecuali *leaf*) hanya memiliki satu *child*.



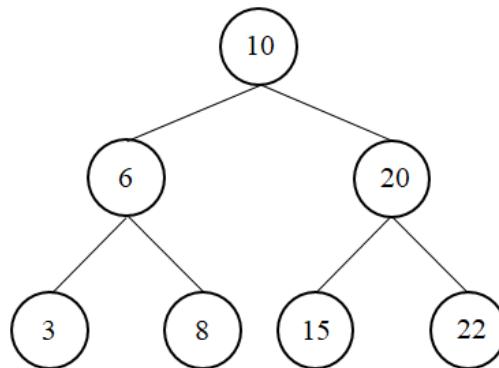
Gambar 7.10 Skewed Binary Tree

## F. Binary Search Tree

*Binary search tree* adalah *Binary Tree* dengan sifat bahwa semua *left child* harus lebih kecil daripada *right child* dan *parentnya*. Juga semua *right child* harus lebih besar dari *left child* serta *parentnya*. *Binary search tree* dibuat untuk mengatasi kelemahan pada *binary tree* biasa, yaitu kesulitan dalam *searching* / pencarian *node* tertentu dalam *binary tree*.

Contoh inputan nilai : 10, 6, 20, 3, 8, 15, 22

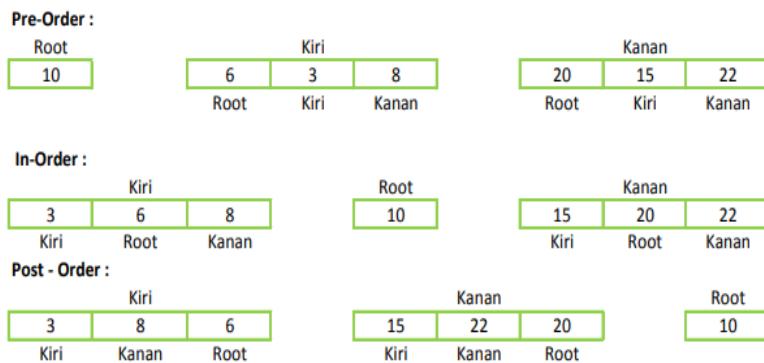
Nilai yang lebih kecil dari parent berada di sebelah kiri dan nilai yang lebih besar di sebelah kanan.



Gambar 7.11 Binary Search Tree

Transversing :

- 1) *Pre-order* : *root – kiri – kanan*
- 2) *In-order* : *kiri – root – kanan*
- 3) *Post-order* : *kiri – kanan – root*



Gambar 7.12 Transversing

## G. Operasi – operasi pada Tree

### 1) Create

Membentuk *binary tree* baru yang masih kosong.

### 2) Clear

Mengosongkan *binary tree* yang sudah ada.

### 3) Empty

*Function* untuk memeriksa apakah *binary tree* masih kosong.

### 4) Insert

Memasukkan sebuah *node* ke dalam *tree*. Ada tiga pilihan *insert*: sebagai *root*, *left child*, atau *right child*. Khusus *insert* sebagai *root*, *tree* harus dalam keadaan kosong.

### 5) Find

Mencari *root*, *parent*, *left child*, atau *right child* dari suatu *node*. (*Tree* tak boleh kosong).

### 6) Update

Mengubah isi dari *node* yang ditunjuk oleh *pointer current*. (*Tree* tidak boleh kosong).

### 7) Retrieve

Mengetahui isi dari *node* yang ditunjuk *pointer current*. (*Tree* tidak boleh kosong).

### 8) DeleteSub

Menghapus sebuah *subtree* (*node* beserta seluruh *descendantnya*) yang ditunjuk *current*. *Tree* tak boleh kosong. Setelah itu *pointer current* akan berpindah ke *parent* dari *node* yang dihapus.

9) *Characteristic*

Mengetahui karakteristik dari suatu *tree*, yakni : *size*, *height*, serta *average lengthnya*. *Tree* tidak boleh kosong

10) *Traverse*

Mengunjungi seluruh *node-node* pada *tree*, masing-masing sekali. Hasilnya adalah urutan informasi secara linier yang tersimpan dalam *tree*.

## H. Pembuatan Tree

Pembuatan *tree* lebih mudah menggunakan *binary search tree* dimana jika nilai simpul lebih kecil dari *parent* maka akan ditempatkan pada bagian kiri dan jika nilai simpul lebih besar akan ditempatkan di sebelah kanan.

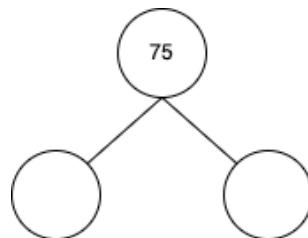
### 7.5 Tugas Praktikum ke-1: Membuat Tree

A. Urutkan secara Ascending :

Pada ascending jika data

1) Proses 1

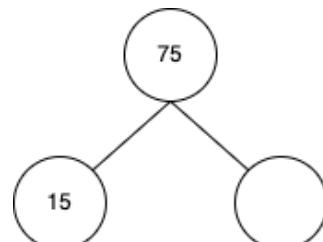
Root = 75



Gambar 7.13 Pemasukan Data Pertama Ke Root

2) Proses 2

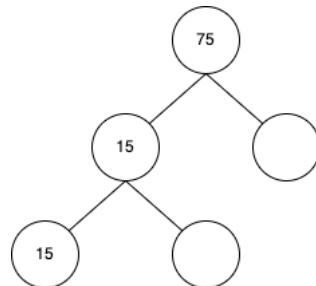
Cek apakah  $15 < 75$ ? Karena  $15 < 75$ , maka diletakkan disebelah kiri root.



Gambar 7.14 Pembandingan Data Pertama

3) Proses 3

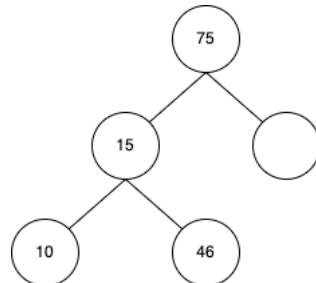
Cek apakah  $10 < 75$ ? Karena  $10 < 75$ , maka diletakkan disebelah kiri Root, selanjutnya dicek lagi, apakah  $10 < 15$ ? Karena  $10 < 15$ , maka diletakkan disebelah kiri.



Gambar 7.15 Pembandingan Data

4) Proses 4

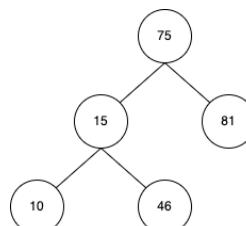
Cek apakah  $46 < 75$ ? Karena  $46 < 75$ , maka diletakkan disebelah kiri Root, selanjutnya dicek lagi, apakah  $46 < 15$ ? Karena  $46 > 15$ , maka diletakkan disebelah kanan.



Gambar 7.16 Pemasukan Data

5) Proses 5

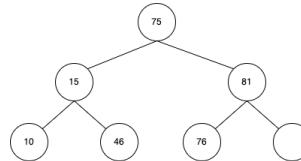
Cek apakah  $81 < 75$ ? Karena  $81 > 75$ , maka diletakkan disebelah kanan Root.



Gambar 7.17 Memasukan Data Sub Tree Kanan

6) Proses 6

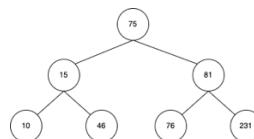
Cek apakah  $76 < 75$ ? Karena  $76 > 75$ , maka diletakkan disebelah kanan Root, selanjutnya dicek lagi, apakah  $76 < 81$ ? Karena  $76 < 81$ , maka diletakkan disebelah kiri.



Gambar 7.18 Memasukan Data Sub Tree Kanan

7) Proses 7

Cek apakah  $231 < 75$ ? Karena  $231 > 75$ , maka diletakkan disebelah kanan Root, selanjutnya dicek lagi, apakah  $231 < 81$ ? Karena  $231 > 81$ , maka diletakkan disebelah kanan.

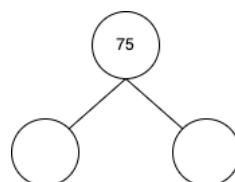


Gambar 7.19 Memasukan Data Sub Tree Kanan

B. Secara Descending

1) Proses 1

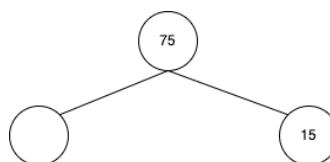
Root=75



Gambar 7.20 Memasukan Data Awal

2) Proses 2

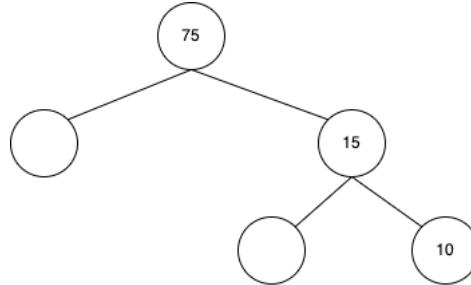
Cek apakah  $15 < 75$ ? Karena  $15 < 75$ , maka diletakkan disebelah kanan root



Gambar 7.21 Memasukan Data Kanan

3) Proses 3

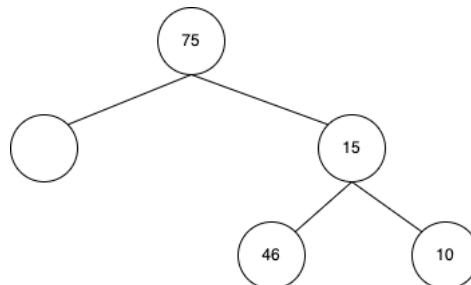
Cek apakah  $10 \leftrightarrow 75$ ? Karena  $10 < 75$ , maka diletakkan disebelah kanan Root, selanjutnya dicek lagi, apakah  $10 \leftrightarrow 15$ ? Karena  $10 < 15$ , maka diletakkan disebelah kanan.



Gambar 7.22 Memasukan Sub Data Kanan

4) Proses 4

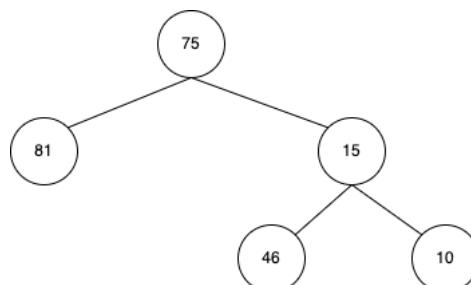
Cek apakah  $46 \leftrightarrow 75$ ? Karena  $46 < 75$ , maka diletakkan disebelah kanan Root, selanjutnya dicek lagi, apakah  $46 \leftrightarrow 15$ ? Karena  $46 > 15$ , maka diletakkan disebelah kiri.



Gambar 7.23 Memasukan Data

5) Proses 5

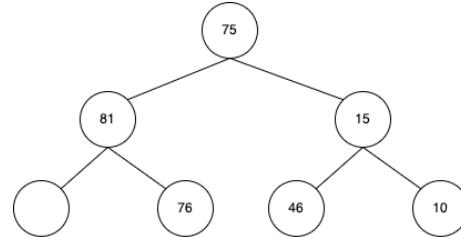
Cek apakah  $81 \leftrightarrow 75$ ? Karena  $81 > 75$ , maka diletakkan disebelah kiri Root.



Gambar 7.24 Memasukan Sub Data Kanan

### 6) Proses 6

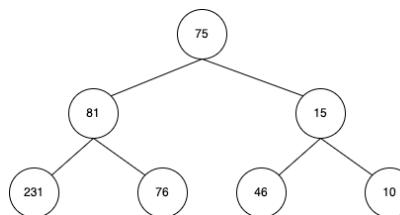
Cek apakah  $76 \leftrightarrow 75$ ? Karena  $76 > 75$ , maka diletakkan disebelah kiri Root, selanjutnya dicek lagi, apakah  $76 \leftrightarrow 81$ ? Karena  $76 < 81$ , maka diletakkan disebelah kanan.



Gambar 7.25 Memasukan Sub Data Kiri

### 7) Proses 7

Cek apakah  $160 \leftrightarrow 75$ ? Karena  $160 > 75$ , maka diletakkan disebelah kiri Root, selanjutnya dicek lagi, apakah  $160 \leftrightarrow 81$ ? Karena  $160 > 81$ , maka diletakkan disebelah kiri.



Gambar 7.26 Memasukan Sub Data Kiri

Tabel 7.1 *Transversin Table ascending :*

IN-order :	kiri			Root	Kanan		
	10	15	45		76	81	231
	Kiri	Root	Kanan		kiri	Root	Kanan

Pre-order :	Root	Kiri			Kanan		
	75	15	10	45	81	76	231
	Root	Root	kiri	kanan	Root	kiri	Kanan

Post-order :	kiri			kanan			root
	10	45	15	76	231	81	75
	Kiri	Kanan	Root	kiri	Kanan	Root	Root

Tabel 7.2 *Transversing Table descending :*

IN-order :	kiri			Root	Kanan		
	231	81	76	75	46	15	10
	Kiri	Root	Kanan	Root	kiri	root	Kanan

Pre-order :	Root	Kiri			Kanan		
	75	81	231	76	15	46	10
	Root	Root	kiri	kanan	Root	kiri	Kanan

Post-order :	kiri			kanan			root
	231	76	81	46	10	15	75
	Kiri	Kanan	root	kiri	Kanan	root	root

## 7.6 Tugas Praktikum ke-2: Program Tree

*Source Code :*

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

struct Node{
    int data;
    Node *kiri;
    Node *kanan;
};

Node *pohon = NULL;

void tampil(Node *root){
    if(root != NULL){
        printf("%d ", root->data);
    }
}
```

```

        tampil(root->kiri);
        tampil(root->kanan);
    }
}

void tambah(Node **root, int databaru){
    if((*root) == NULL){
        Node *baru;
        baru = new Node;
        baru->data = databaru;
        baru->kiri = NULL;
        baru->kanan = NULL;
        (*root) = baru;
        (*root)->kiri = NULL;
        (*root)->kanan = NULL;
        printf("Data bertambah!");
    }else if(databaru<(*root)->data)
        tambah(&(*root)->kiri, databaru);
    else if(databaru>(*root)->data)
        tambah(&(*root)->kanan, databaru);
    else if(databaru == (*root)->data)
        printf("Data sudah ada!");
}
void preOrder(Node *root){
    if(root != NULL){
        printf("%d ", root->data);
        preOrder(root->kiri);
        preOrder(root->kanan);
    }
}
void inOrder(Node *root){
    if(root != NULL){
        inOrder(root->kiri);
        printf("%d ", root->data);
        inOrder(root->kanan);
    }
}
void postOrder(Node *root){
    if(root != NULL){
        postOrder(root->kiri);
        postOrder(root->kanan);
        printf("%d ", root->data);
    }
}
int height(Node *root)
{
    if(root == NULL)
        return -1;
    else{
        int u = height(root->kiri);
        int v = height(root->kanan);
        if(u > v)
            return u + 1;
        else
            return v + 1;
    }
}

```

```

void clear(Node **root, int del)
{
    Node *curr;
    Node *parent;
    curr = (*root);

    bool flag = false;

    while(curr != NULL)
    {
        if(curr->data == del)
        {
            flag = true;
            break;
        }
        else
        {
            parent = curr;
            if(del>curr->data)
                curr = curr->kanan;
            else
                curr = curr->kiri;
        }
    }

    if(!flag)
    {
        cout<<"Data tidak ditemukan. Penghapusan tidak dilakukan."<<endl;
        return;
    }
    if(height(pohon) == 0)
    {
        if( curr->kiri== NULL && curr->kanan == NULL)
        {
            (*root) = NULL;

            return;
        }
    }
    else if(height(pohon) > 0)
    {
        if( curr->kiri== NULL && curr->kanan == NULL)
        {
            if(parent->kiri == curr)
            {
                parent->kiri = NULL;
                delete curr;
            }
            else
            {
                parent->kanan = NULL;
                delete curr;
            }
        }

        return;
    }
    if((curr->kiri == NULL && curr->kanan != NULL) || (curr->kiri != NULL && curr->kanan == NULL))

```

```

    {
        if(curr->kiri == NULL && curr->kanan != NULL)
        {
            if(parent->kiri == curr)
            {
                parent->kiri = curr->kanan;
                delete curr;
            }
            else
            {
                parent->kanan = curr->kanan;
                delete curr;
            }
        }
        else
        {
            if(parent->kiri == curr)
            {
                parent->kiri = curr->kiri;
                delete curr;
            }
            else
            {
                parent->kanan = curr->kiri;
                delete curr;
            }
        }
        return;
    }
    if (curr->kiri != NULL && curr->kanan != NULL)
    {
        Node* bantu;
        bantu = curr->kanan;
        if((bantu->kiri == NULL) && (bantu->kanan == NULL))
        {
            curr = bantu;
            delete bantu;
            curr->kanan = NULL;
        }
        else
        {

            if((curr->kanan)->kiri != NULL)
            {
                Node* bantu2;
                Node* bantu3;
                bantu3 = curr->kanan;
                bantu2 = (curr->kanan)->kiri;
                while(bantu2->kiri != NULL)
                {
                    bantu3 = bantu2;
                    bantu2 = bantu2->kiri;
                }
                curr->data = bantu2->data;
                delete bantu2;
            }
        }
    }
}

```

```

        bantu3->kiri = NULL;
    }
    else
    {
        Node* tmp;
        tmp = curr->kanan;
        curr->data = tmp->data;
        curr->kanan = tmp->kanan;
        delete tmp;
    }

}
return;
}

}

int main(){
    int pil, data,del;

    do{
        system("cls");
        printf("\n2118131 | Moh harifin\n");
        printf("\t#PROGRAM TREE C++#");
        printf("\n\t=====");
        printf("\nMENU");
        printf("\n----\n");
        printf("1. Tambah\n");
        printf("2. Lihat pre-order\n");
        printf("3. Lihat in-order\n");
        printf("4. Lihat post-order\n");
        printf("5. Clear Data\n");
        printf("6. Exit\n");
        printf("Pilihan : ");
        scanf("%d", &pil);
        switch(pil){
        case 1 :
            printf("\nINPUT : ");
            printf("\n----");
            printf("\nData baru : ");
            scanf("%d", &data);
            tambah(&pohon, data);
            break;
        case 2 :
            printf("\nData yang tersedia : ");
            if(pohon!=NULL)
                preOrder(pohon);
            else
                printf("Masih kosong!");
            printf("\nOUTPUT PRE ORDER : ");
            if(pohon!=NULL)
                preOrder(pohon);
            else
                printf("Masih kosong!");
            break;
        case 3 :
            printf("\nData yang tersedia : ");
            if(pohon!=NULL)

```

```

        preOrder(pohon);
    else
        printf("Masih kosong!");

    printf("\nOUTPUT IN ORDER : ");
    if(pohon!=NULL)
        inOrder(pohon);
    else
        printf("Masih kosong!");
break;
case 4 :
    printf("\nData yang tersedia : ");
    if(pohon!=NULL)
        preOrder(pohon);
    else
        printf("Masih kosong!");

    printf("\nOUTPUT POST ORDER : ");
    if(pohon!=NULL)
        postOrder(pohon);
    else
        printf("Masih kosong!");
break;

case 5:
    if(pohon != NULL) {
        printf("Hapus data: ");
        scanf("%d", &del);
        clear(&pohon, del);
        printf("Data telah terhapus!");

        printf("\n\nSETELAH NODE DIHAPUS : ");
        printf("-----\n");
        printf("\nPRE ORDER : ");
        preOrder(pohon);
        printf("\nIN ORDER : ");
        inOrder(pohon);
        printf("\nPOST ORDER : ");
        postOrder(pohon);
    }else
        printf("Masih kosong!");
    _getch();
break;

} _getch();
}while(pil != 6);
return EXIT_FAILURE;
}

```

Tampilan program :

```
2118131 : Moh harifin
#PROGRAM TREE C++#
=====
MENU
---
1. Tambah
2. Lihat pre-order
3. Lihat in-order
4. Lihat post-order
5. Clear Data
6. Exit
Pilihan : 2

Data yang tersedia : 75 15 10 46 81 76 231
OUTPUT PRE ORDER : 75 15 10 46 81 76 231
```

Gambar 7.27 Hasil tampilan tree

Analisa Program :

Program diatas menjelaskan tentang alur dari materi *Tree*. Jika, *user* memilih pertama maka akan melakukan penambahan data baru. Dan akan dilakukan pemanggilan function tambah(&pohon,data) untuk melakukan penambahan. Di dalam statement function tambah tersebut terdapat statement dengan menggunakan kondisi if ((\*root)==NULL) jika root kosong maka menjalankan statement di dalamnya tapi jika tidak kosong maka akan menjalankan program berikutnya. Jika user memilih dua maka akan menampilkan data yang sudah di inputkan secara pre-order begitu juga dengan pilihan tiga dan empat. Dan akan dilakukan pemanggilan sesuai ketentuan masing masing yang sudah di tentukan dan akan di lakukan pemanggilan function untuk menampilkannya. Jika user memilih tiga maka user akan menginputkan data man yang ingin di hapus. Jika sudah maka akan melakukan pemanggilan funtion dengan ketentuan yang sudah di tentukan. Jika user memilih angka terakhir maka program selesai.

## 7.7 Kesimpulan

1. *Tree* merupakan salah satu bentuk struktur data tidak linier yang menggambarkan hubungan yang bersifat hierarkis
2. *Tree* dibedakan menjadi beberapa jenis, antara lain, pohon keluarga dan *pare tree*,
3. *Binary search tree* dibagi menjadi beberapa jenis, yaitu *full binary tree*, *complete binary tree*, dan *skewed binary tree*

Disetujui Aslab Alfina (2018041) Tgl:	Ttd / Paraf
---	-------------

## KESIMPULAN

1. *Struct* dapat diartikan sebagai penggabungan dari beberapa elemen menjadi satu kesatuan
2. Pengalaman Memori dengan *pointer* dapat berbeda beda, karena tiap memori dalam laptop memiliki alamat yang berbeda-beda.
3. Program *rekursif* dapat digunakan untuk menyelesaikan program permutasi dan program sejenisnya.
4. *Stack* dapat diartikan sebagai antrian, dimana antrian ini memiliki kepala dan ekor antrian.
5. Metode Penyelesaian *stack* dibagi menjadi beberapa macam, seperti *Enqueue*, *Dequeue*, *Is Full*, *IsEmpty*, *Count*, *Display*, dan *Destroy*
6. *Sorting* berfungsi sebagai program pengurutan data yang tidak urut baik secara *ascending* maupun *descending*
7. Programs *sorting* memiliki beberapa metode, seperti *Bubble*, *Exchange*, *Insertion*, *Selection*, *Insection*, dan *Quick Sorting*
8. Metode *Searching* merupakan metode untuk mencari suatu data tertentu secara acak.
9. *Linked list* dapat diartikan sekumpulan elemen dari data yang bertipe sama yang saling terurut dan terhubung dengan bantuan variabel *pointer*
10. *Linked list* adalah sekumpulan elemen dari data yang bertipe sama yang saling terurut dan terhubung dengan bantuan variabel *pointer*, yang setiap data di dalamnya disebut *node* ( simpul ) menempati alokasi memori secara dinamis dan biasanya berupa *struct* yang terdiri dari beberapa *field*.
11. *Single linked list* merupakan *linked list* yang *field pointer*-nya hanya satu buah saja dan satu arah.
12. *Double linked list* adalah *linked list* yang *field pointer*nya 2 arah. Merujuk *node* sebelum dan sesudahnya.
13. *Tree* merupakan merupakan salah satu bentuk struktur data tidak linier yang menggambarkan hubungan yang bersifat hierarkis
14. Dalam *Linked List* datanya memiliki *head* dan *tail* untuk memudahkan pemasukan atau penghapusan data

## **DAFTAR PUSTAKA**

- Jaringan komputer, Laboratorium. 2022. *Modul Struktur Data*. Malang: Netlab ITN Malang.
- Lenti, Febri Nova. "LIST BERKAIT DENGAN DEFENISI DAN PROSES REKURSIF." Majalah Ilmiah Format 7.1 (2006): 835-863.
- Torre, Salvatore La, Parthasarathy Madhusudan, and Gennaro Parlato. "Context-bounded analysis of concurrent queue systems." International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2008.
- Al-Rashed, Rwan F., Et Al. "EVALUATING THE REVERSING DATA STRUCTURE ALGORITHM OF LINKED LIST."
- Pradana, Ragel Wira Agung. Pembangunan Materi Digital Untuk Konsep Searching Pada Media Pembelajaran Algoritma. Diss. Fakultas Teknik, 2018.
- Booch, G., & Vilot, M. (1990, September). The design of the C++ Booch components. In Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications (pp. 1-11).



LABORATORIUM JARINGAN KOMPUTER  
INSTITUT TEKNOLOGI NASIONAL  
Kampus II : Jl. Raya Karanglo Km. 2 Malang

**LEMBAR ASISTENSI PRAKTIKUM STRUKTUR DATA**  
**SEMESTER GENAP TAHUN AKADEMIK 2021/2022**

Nama : Moh harifin

NIM : 2118131

**FOTO**  
**3X4**  
**BG**

No.	Tanggal	Asistensi			Paraf
		Konsep	Program	Hasil Akhir	
1		Instruktur	<input type="checkbox"/> Pointer, Struct, Rekursif <input type="checkbox"/> Queue, Stack <input type="checkbox"/> Sorting <input type="checkbox"/> Searching <input type="checkbox"/> Linked List <input type="checkbox"/> Tree		
2		Dosen			
<b>Batas Akhir ACC Aslab: 18 Juni 2022</b>					

Asisten,

(Alfina)

2018041

**Malang, Juni 2022**  
Dosen, Pembimbing,

**Dedy Rudhistiar, S.Kom.M.Cs**  
**NIP. P. 1032000578**