

ALGORITMA PEMROGRAMAN TERSTRUKTUR



MUHAMMAD BAHIT

ALGORITMA PEMROGRAMAN TERSTRUKTUR

Undang-Undang No. 28 Tahun 2014 Tentang Hak Cipta

Fungsi dan sifat hak cipta Pasal 4

Hak Cipta sebagaimana dimaksud dalam Pasal 3 huruf a merupakan hak eksklusif yang terdiri atas hak moral dan hak ekonomi.

Pembatasan Perlindungan Pasal 26

Ketentuan sebagaimana dimaksud dalam Pasal 23, Pasal 24, dan Pasal 25 tidak berlaku terhadap :

- i. penggunaan kutipan singkat Ciptaan dan/atau produk Hak Terkait untuk pelaporan peristiwa aktual yang ditujukan hanya untuk keperluan penyediaan informasi aktual;
- ii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk kepentingan penelitian ilmu pengetahuan;
- iii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk keperluan pengajaran, kecuali pertunjukan dan Fonogram yang telah dilakukan Pengumuman sebagai bahan ajar; dan
- iv. penggunaan untuk kepentingan pendidikan dan pengembangan ilmu pengetahuan yang memungkinkan suatu Ciptaan dan/atau produk Hak Terkait dapat digunakan tanpa izin Pelaku Pertunjukan, Produser Fonogram, atau Lembaga Penyiaran.

Sanksi Pelanggaran Pasal 113

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp 100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp 500.000.000,00 (lima ratus juta rupiah).
3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp 1.000.000.000,00 (satu miliar rupiah).
4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp 4.000.000.000,00 (empat miliar rupiah).

ALGORITMA PEMROGRAMAN TERSTRUKTUR

Muhammad Bahit



Poliban Press

Algoritma Pemrograman Terstruktur

Penulis :

Muhammad Bahit

ISBN :

978-623-5259-10-9 (PDF)

Editor dan Penyunting :

Reza Fauzan

Desain Sampul dan Tata letak :

Eko Sabar Prihatin; Rahma Indera

Penerbit :

POLIBAN PRESS

Anggota APPTI (Asosiasi Penerbit Perguruan Tinggi Indonesia)

no.004.098.1.06.2019

Cetakan Pertama, 2024

Hak cipta dilindungi undang-undang

Dilarang memperbanyak karya tulis ini dalam bentuk
dan dengan cara apapun tanpa ijin tertulis dari penerbit

Redaksi :

Politeknik Negeri Banjarmasin, Jl. Brigjen H. Hasan Basry,
Pangeran, Komp. Kampus ULM, Banjarmasin Utara

Telp : (0511)3305052

Email : press@poliban.ac.id

Diterbitkan pertama kali oleh :

Poliban Press, Banjarmasin, Januari 2024

KATA PENGANTAR

Buku ini merupakan panduan menyeluruh dalam memahami dan mengimplementasikan konsep-konsep penting dalam pemrograman komputer. Dari Tipe Data & Sorting hingga Binary Tree dan Hash Table, setiap bab memberikan penjelasan teori yang ringkas dan praktek yang mendalam.

Buku ini dimulai dengan pembahasan mengenai tipe data dasar seperti Integer, Floating Point, Character, String, Boolean, dan Enum. Setelah itu, Anda akan dihadapkan pada konsep Array, Pointer, dan Referensi, diikuti oleh praktik nyata dalam membuat program sederhana seperti "Hello World" dan manipulasi larik.

Buku ini dirancang agar mudah dipahami dengan struktur yang terorganisir, dilengkapi dengan praktik nyata, sehingga pembaca dapat membangun dasar yang kuat dalam pemrograman C++. Kami berharap buku ini memberikan kontribusi positif dalam perjalanan pembaca dalam memahami dan menguasai konsep-konsep dasar pemrograman. Selamat membaca dan semoga sukses dalam eksplorasi dunia pemrograman!

Banjarmasin, Desember 2023

Penerbit

PRAKATA

Assalamu'alaikum wa rahmatullahi wa barakatuh.

Puji syukur alhamdulillah kami panjatkan ke hadirat Tuhan Yang Maha Esa yang telah melimpahkan nikmat, taufik serta hidayah-Nya yang sangat besar sehingga Bahan Ajar Algoritma Pemrograman Terstruktur dapat diselesaikan.

Mudah-mudahan buku ajar ini bisa membantu mahasiswa untuk memahami secara kontekstual. Penulis yakin bahwa materi dalam bahan kuliah ini masih jauh dari kesempurnaan, sehingga terbuka untuk mendapatkan kritik dan saran untuk perbaikan pada semua sisi penulisannya.

Wassalamu'alaikum wa rahmatullahi wa barakatuh

Banjarmasin, Desember 2023

Muhammad Bahit

DAFTAR ISI

KATA PENGANTAR.....	v
PRAKATA.....	vi
DAFTAR ISI	vii
DAFTAR GAMBAR.....	x
DAFTAR TABEL.....	xii
BAB I TIPE DATA & SORTING.....	1
1.1. Pengertian Tipe Data	1
A. Integer.....	2
B. Floating Point (float)	3
C. Character (char)	5
D. String	5
E. Boolean.....	6
F. Enum	7
1.2. Array (larik).....	8
1.3. Pointer	9
1.4. Referensi.....	9
1.5. Praktik Hello Word.....	9
1.6. Praktik Menampilkan Nama.....	10
1.7. Praktik Larik.....	11
1.8. Sorting	14
A. Selection Sort	14
B. Insertion Sort	15
1.9. Praktik Sorting.....	15
1.10. Soal Latihan	19
BAB II MERGE SORT, REKURSIF & QUICK SORT	20
2.1. Teori Singkat Merge Sort	20
2.2. Praktik Merge Sort	21
2.3. Teori Singkat Rekursif.....	24
2.4. Teori Singkat Quick Sort.....	24
2.5. Bubble Sort.....	26
2.6. Praktik Rekursif & Quick Sort	26

BAB III SEARCHING & STACK	31
3.1. Teori Singkat Searching	31
A. Pencarian sekuensial (<i>sequential search</i>)	31
B. Pencarian Biner (<i>Binary Search</i>)	32
3.2. Praktik Searching	32
3.3. Teori Singkat Stack	36
3.4. Praktik Stack.....	37
3.5. Soal Latihan.....	39
BAB IV QUEUE & LINK LIST	40
4.1. Teori Singkat Queue.....	40
4.2. Praktik QUEUE.....	41
4.3. Teori Singkat Link List 1	44
4.4. Praktik Link List Penambahan Simpul Di Depan Dan Di Belakang.....	45
4.5. Teori Singkat Link List 2	48
4.6. Praktik Link List Menambah Data Di Posisi Tertentu Dan Menghapus Data.....	48
4.7. Soal Latihan.....	54
BAB V DOUBLE LINK LIST.....	55
4.1 Teori Singkat Double Link list (Penambahan Simpul Didepan).....	55
4.2 Praktik Penambahan Simpul Didepan.....	58
4.3 Teori Singkat Double Link list (Penambahan Simpul Ditengah).....	63
4.4 Praktik Penambahan Simpul Ditengah.....	64
4.5 Soal Latihan	70
BAB VI BINARY TREE.....	71
6.1. Teori Singkat Binary Tree.....	71
6.2. Praktik Binary Tree.....	72
6.3. Teori Singkat Binary Tree (Sesi 2)	83
6.4. Praktik Binary Tree (Sesi 2)	84
6.5. Soal Latihan.....	87
BAB VII HASH TABLE.....	88
7.1. Teori Singkat Hash Table.....	88
7.2. Memilih Fungsi Hash	89

7.3. Menangani Tabrakan (<i>Collision</i>) Dalam Tabel Hash	90
7.4. Praktik Hast Table	91
7.5. Soal Latihan.....	97
DAFTAR PUSTAKA	98

DAFTAR GAMBAR

Gambar 1.1 Output Kode Program Integer	3
Gambar 1.2 Output Kode Program Floating	4
Gambar 1.3 Output Kode Program Character	5
Gambar 1.4 Output Kode Program String	6
Gambar 1.5 Output Kode Program Boolean	7
Gambar 1.6 Output Kode Program Enum	8
Gambar 1.7 Output Kode Program Hello Word	10
Gambar 1.8 Output Kode Program Menampilkan Nama	11
Gambar 1.9 Output Kode Program Larik	12
Gambar 1.10 Output Kode Program.....	12
Gambar 1.11 Output Kode Program.....	14
Gambar 1.12 Output Kode Program Sorting	17
Gambar 2.1 Output Kode Program Merge Sort.....	24
Gambar 2.2 Output Kode Program Quick Sort	27
Gambar 2.3 Output Kode Program Quick Sort	30
Gambar 3.1 Output Program Searching	35
Gambar 3.2 Output Program Searching	36
Gambar 4. 1 Output Kode Program Queue	43
Gambar 4.2 Merupakan ilustrasi penambahan data di belakang link list.....	45
Gambar 4.3 Ilustrasi penambahan data di posisi tertentu	48
Gambar 4.4 Ilustrasi penghapusan data di posisi depan	48
Gambar 4.5 Output kode program link list.....	54
Gambar 5.1 Ilustrasi Sebuah Penunjuk atau <i>Pointer</i>	55
Gambar 5.2 Ilustrasi Sebuah Simpul dalam <i>Double Linked List</i>	56
Gambar 5.3 Ilustrasi Sebuah <i>Double Linked List</i>	57
Gambar 5.4 Ilustrasi penambahan data di belakang pada <i>double link</i> <i>list</i>	58
Gambar 5.5 Output program penambahan simpul.....	63

Gambar 5.6 Ilustrasi Penambahan Data di Tengah pada *Double Link*

List 63

Gambar 5.7 Ilustrasi Penghapusan Data pada *Double Link List* 64

Gambar 6.1 Output program Binary Tree..... 79

DAFTAR TABEL

Tabel 1. 1 Contoh Larik..... 8

BAB I

TIPE DATA & SORTING

Capaian Pembelajaran

1. Memahami tentang penggunaan tipe data larik dan pointer.
2. Menggunakan tipe data untuk menyelesaikan soal.
3. Memahami tentang sorting dengan metode insertion sort dan selection sort.
4. Menggunakan sorting tersebut dalam implementasi program.

1.1. Pengertian Tipe Data

Tipe data adalah format penyimpanan data yang bisa berisi tipe atau rentang nilai tertentu. Ketika program komputer menyimpan data dalam variabel, setiap variabel harus diberi tipe data tertentu. Beberapa tipe data umum termasuk bilangan bulat, angka floating point, karakter, string, dan array. Mereka mungkin juga tipe yang lebih spesifik, seperti tanggal, stempel waktu, nilai boolean, dan format varchar (karakter variabel).

Beberapa bahasa pemrograman mengharuskan pemrogram untuk menentukan tipe data dari variabel sebelum memberikan nilai. Bahasa lain dapat secara otomatis menetapkan tipe data variabel ketika data awal dimasukkan ke dalam variabel. Misalnya, jika variabel "var1" dibuat dengan nilai "1,25", variabel tersebut akan dibuat sebagai tipe data titik mengambang. Jika variabel disetel ke "Halo!," variabel akan diberi tipe data string. Sebagian besar bahasa pemrograman mengizinkan setiap variabel untuk menyimpan satu tipe data. Oleh karena itu, jika tipe data variabel telah disetel ke bilangan bulat, menetapkan data string ke variabel dapat menyebabkan data diubah ke format bilangan bulat.

Tipe data juga digunakan oleh aplikasi database. Bidang dalam database sering kali membutuhkan tipe data tertentu untuk dimasukkan. Misalnya, catatan perusahaan untuk seorang karyawan dapat menggunakan tipe data string untuk nama depan dan belakang karyawan tersebut. Tanggal perekrutan karyawan akan disimpan dalam format tanggal, sedangkan gajinya dapat disimpan sebagai bilangan bulat. Dengan menjaga tipe data seragam di beberapa catatan, aplikasi database dapat dengan mudah mencari, mengurutkan, dan membandingkan bidang dalam catatan yang berbeda. Beberapa tipe data secara umum sebagai berikut;

A. Integer

Tipe data INTEGER menyimpan bilangan bulat yang berkisar dari -2.147.483.647 hingga 2.147.483.647 untuk presisi 9 atau 10 digit. Angka 2.147.483.648 adalah nilai yang dicadangkan dan tidak dapat digunakan. Nilai INTEGER disimpan sebagai bilangan bulat biner bertanda dan biasanya digunakan untuk menyimpan jumlah, kuantitas, dan sebagainya. Operasi aritmatika dan perbandingan pengurutan dilakukan lebih efisien pada data integer daripada pada data float atau desimal. Namun, kolom INTEGER tidak dapat menyimpan nilai absolut di luar (231-1). Jika nilai data berada di luar rentang numerik INTEGER, server database tidak menyimpan nilai tersebut. Tipe data INTEGER membutuhkan 4 byte penyimpanan per nilai. Tuliskan kode program dibawah ini, jalankan dan analisis hasilnya.

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

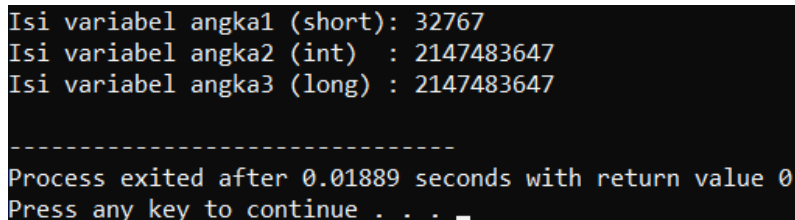
int main()
{
    short angka1;
    int  angka2;
    long angka3;
```

```
angka1 = 32767;
angka2 = 2147483647;
angka3 = 2147483647;

cout << "Isi variabel angka1 (short): " << angka1 << endl;
cout << "Isi variabel angka2 (int) : " << angka2 << endl;
cout << "Isi variabel angka3 (long) : " << angka3 << endl;

return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Isi variabel angka1 (short): 32767
Isi variabel angka2 (int) : 2147483647
Isi variabel angka3 (long) : 2147483647
-----
Process exited after 0.01889 seconds with return value 0
Press any key to continue . . . █
```

Gambar 1. 1 Output Kode Program Integer

B. Floating Point (float)

Jika kita menggunakan 197.0 literal dalam sebuah program, titik desimal memberitahu kompiler untuk mewakili nilai menggunakan tipe data primitif floating point. Pola bit yang digunakan untuk floating point 197.0 sangat berbeda dengan yang digunakan untuk bilangan bulat 197. Ada dua tipe primitif floating point. Float tipe data kadang-kadang disebut "titik mengambang presisi tunggal". Tipe data ganda memiliki bit dua kali lebih banyak dan kadang-kadang disebut "titik mengambang presisi ganda". Frasa ini berasal dari bahasa FORTRAN,

yang pada suatu waktu merupakan bahasa pemrograman yang dominan. Dalam program sumber, literal floating point selalu memiliki titik desimal di dalamnya, dan tidak ada koma (tidak ada pemisah ribuan). Tuliskan kode program dibawah ini, jalankan dan analisis hasilnya.

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

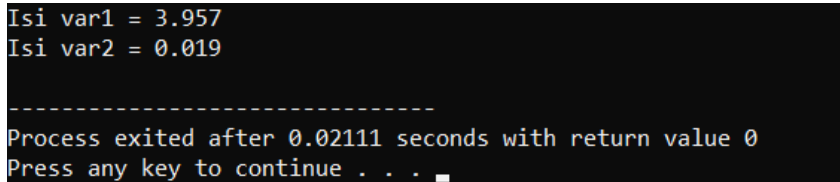
int main()
{
    float var1;
    double var2;

    var1 = 3.957;
    var2 = 0.019;

    cout << "Isi var1 = " << var1 << endl;
    cout << "Isi var2 = " << var2 << endl;

    return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Isi var1 = 3.957
Isi var2 = 0.019
-----
Process exited after 0.02111 seconds with return value 0
Press any key to continue . . . █
```

Gambar 1. 2 Output Kode Program Floating

C. Character (char)

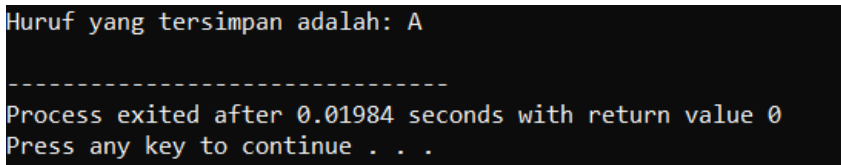
Tipe data char digunakan untuk menyimpan satu karakter. Karakter harus diapit oleh tanda kutip tunggal, seperti 'A' atau 'c'. Tuliskan kode program dibawah ini, jalankan dan analisis hasilnya.

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

int main()
{
    const char huruf = 'A';
    cout << "Huruf yang tersimpan adalah: " << huruf << endl;

    return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Huruf yang tersimpan adalah: A
-----
Process exited after 0.01984 seconds with return value 0
Press any key to continue . . .
```

Gambar 1. 3 Output Kode Program Character

D. String

Tipe data *string* secara umum merupakan urutan karakter, baik sebagai konstanta literal atau sebagai semacam variable yang memungkinkan elemen-elemen seperti Panjang dapat diubah, atau mungkin diperbaiki. *String* umumnya dianggap sebagai tipe data dan sering diimplementasikan sebagai struktur data array byte (atau kata) yang menyimpan urutan elemen dengan menggunakan beberapa

pengkodean karakter. Tuliskan kode program dibawah ini, jalankan dan analisis hasilnya.

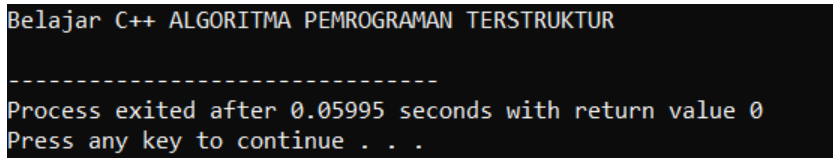
```
#include <iostream>
using namespace std;

int main()
{
    string var1 = "Belajar C++ ";
    string    var2    =    "ALGORITMA    PEMROGRAMAN
    TERSTRUKTUR";

    cout << var1 << var2 << endl;

    return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Belajar C++ ALGORITMA PEMROGRAMAN TERSTRUKTUR
-----
Process exited after 0.05995 seconds with return value 0
Press any key to continue . . .
```

Gambar 1. 4 Output Kode Program String

E. Boolean

Tipe data boolean dideklarasikan dengan kata kunci bool dan hanya dapat mengambil nilai true atau false. Ketika nilai dikembalikan, benar = 1 dan salah = 0. Tuliskan kode program dibawah ini, jalankan dan analisis hasilnya.

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

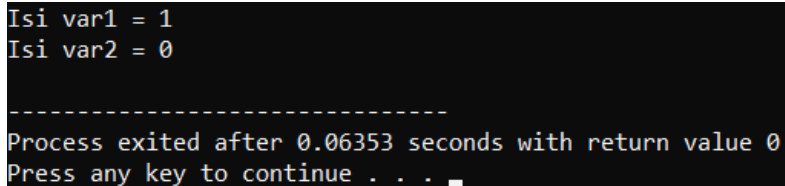
int main()
```

```
{
    bool var1 = true;
    bool var2 = false;

    cout << "Isi var1 = " << var1 << endl;
    cout << "Isi var2 = " << var2 << endl;

    return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Isi var1 = 1
Isi var2 = 0

-----
Process exited after 0.06353 seconds with return value 0
Press any key to continue . . .
```

Gambar 1. 5 Output Kode Program Boolean

F. Enum

Enum adalah objek string dengan nilai yang dipilih dari daftar nilai yang diperbolehkan yang disebutkan secara eksplisit dan spesifikasi. Tuliskan kode program dibawah ini, jalankan dan analisis hasilnya.

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

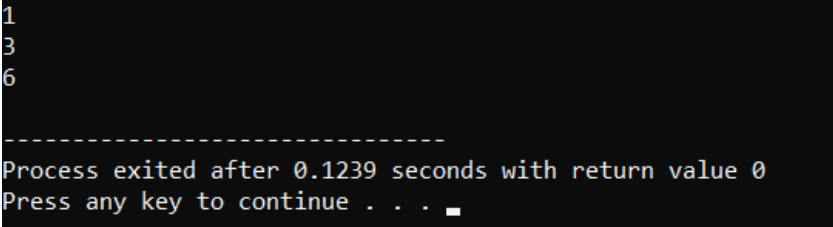
enum nama_hari { minggu, senin, selasa, rabu, kamis, jumat, sabtu };

int main()
{
    nama_hari var1;
    var1 = senin;
    cout << var1 << endl;
}
```

```
var1 = rabu;
cout << var1 << endl;

var1 = sabtu;
cout << var1 << endl;
return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



Gambar 1. 6 Output Kode Program Enum

1.2. Array (larik)

Array atau larik mewakili satu set data. Dalam beberapa bahasa pemrograman, data yang terdapat dalam array harus bertipe sama. Array diwakili oleh awalan huruf kapital, dan notasi [] digunakan untuk mewakili data dalam array. Contoh: Data [12,32,43,47]. Mewakili array data yang berisi data 12, 32, 43, dan 47. Array tersebut dapat dinyatakan dalam bentuk tabel sebagai berikut:

Tabel 1. 1 Contoh Larik

0	1	2	3	← indeks ke
12	32	43	47	Data
data[0]	data[1]	data[2]	data[3]	penyebutan

Array data memiliki 4 elemen setiap elemen diwakili oleh nama array dan indeksny. Indeks dimulai dengan angka 0. Untuk membuat variabel tipe array di C ++ dengan format *tipeData namaLarik[jumlahElemen]*. Contoh data int [4]; Deklarasi mengacu pada deklarasi tipe data array dengan nama data dan tipe integer, dan jumlah elemen adalah 4.

1.3. Pointer

Pointer adalah variabel yang nilainya 0 atau berasal dari variabel atau objek. Jika nilainya 0, itu disebut pointer nol, yang menunjuk ke alamat variabel atau objek yang disimpan dalam memori. *Pointer* yang tidak diinisialisasi disebut dangling pointer dan tidak dapat diprediksi

Salah satu jenis *pointer* adalah "*pointer ke xxxx*", di mana xxxx adalah jenis variabel atau objek yang ditunjuknya. Pengidentifikasi penunjuk adalah xxxx *.

1.4. Referensi

Referensi adalah alias, yang merupakan sinonim untuk variabel atau objek. Jenis yang direferensikan adalah xxxx &, di mana xxxx adalah jenis variabel atau nama objek. Seperti halnya konstanta, referensi harus diinisialisasi. Operator *new* dan *delete new*. Operator *new* membuat objek saat runtime dan membuatnya tetap aktif selama program aktif. Sintaks dari operator *new* adalah xxxx * p = new xxxx (). Di mana xxxx adalah kelas objek. Ekspresi new xxxx() memanggil kelas konstruktor untuk membuat objek baru dan mengembalikan pointer ke sana. Hasilnya adalah objek anonim, diakses oleh *p atau *q, di mana q adalah pointer lain yang sama dengan p. **delete**. Dalam beberapa kasus, perlu untuk menghentikan keberadaan objek sebelum akhir program digunakan operator delete.

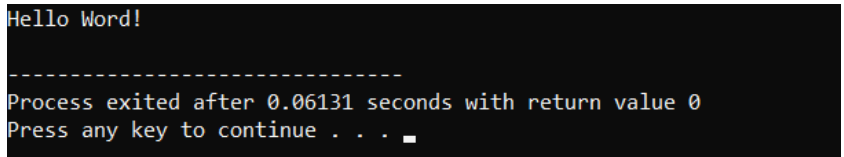
1.5. Praktik Hello Word

Tuliskan kode program dibawah ini, jalankan dan analisis hasilnya.

```
#include <iostream>
int main()
{
    std::cout << "Hello Word! \n";

    return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Hello Word!
-----
Process exited after 0.06131 seconds with return value 0
Press any key to continue . . . ■
```

Gambar 1. 7 Output Kode Program Hello Word

1.6. Praktik Menampilkan Nama

Tuliskan kode program dibawah ini, jalankan dan analisis hasilnya.

```
#include <iostream>
int main()
{
    std::cout << "Hello World! \n";
    std::cout << "SELAMAT BELAJAR DALGORITMA
PEMROGRAMAN TERSTRUKTUR \n";
    std::cout << "Prodi Komputerisasi Akuntansi \n";
    printf("Politeknik Negeri Banjarmasin");

    return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.

```
Muhammad Bahit
SELAMAT BELAJAR DALGORITMA PEMROGRAMAN TERSTRUKTUR
Prodi Komputerisasi Akuntansi
Politeknik Negeri Banjarmasin
-----
Process exited after 0.02416 seconds with return value 0
Press any key to continue . . .
```

Gambar 1. 8 Output Kode Program Menampilkan Nama

1.7. Praktik Larik

1. Cobalah program berikut, jalankan dan analisis hasilnya

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

int main(){

    int bil[5],n,i;
    float total,rata;
    cout<<"Masukan banyak bilangan: "; cin>>n;
    total = 0;
    for(i=1; i<=n; i++){
        cout<<"Bilangan ke "<<i<<" = "; cin>>bil[i];
        total= total + bil[i];
    }
    rata= total/n;
    cout<<"\nTotal bilangan tersebut = "<<total;
    cout<<"\nRata-rata bilangan tersebut = "<<rata;
    getch();
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.


```
Masukan banyak bilangan: 3
Bilangan ke 1 = 2
Bilangan ke 2 = 3
Bilangan ke 3 = 4

Total bilangan tersebut = 9
Rata-rata bilangan tersebut = 3
```

Gambar 1. 9 Output Kode Program Larik

1. Buat sebuah program dengan larik untuk menyimpan nama dan menampilkannya.
2. Cobalah program berikut, jalankan dan analisis hasilnya.
3. Gantilah baris `A = B`; menjadi `A = B`

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

int main(){

    int *A; int p;
    p = 100; A = &p;
    cout<<"Nilai p : "<<p<<endl;
    cout<<"Nilai yang ditunjuk A : "<<*A<<endl;
    cout<<"Alamat yang ditunjuk A : "<<A<<endl;
    getch();
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.

```
Nilai p : 100
Nilai yang ditunjuk A : 100
Alamat yang ditunjuk A : 0x6ffe04
```

Gambar 1. 10 Output Kode Program

1. Selanjutnya ketikkan kode program berikut ini;

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

int main(){

    int *A,*B;
    int p,q;
    p = 100; q = 50; A = &p; B = &q;
    cout<<"Nilai p : "<<p<<endl;
    cout<<"Nilai yang ditunjuk A : "<<*A<<endl;
    cout<<"Alamat yang ditunjuk A : "<<A<<endl;
    cout<<"Nilai q : "<<q<<endl;
    cout<<"Nilai yang ditunjuk B : "<<*B<<endl;
    cout<<"Alamat yang ditunjuk B : "<<B<<endl;
    q = 200;
    *B = *A;
    cout<<"Nilai p : "<<p<<endl;
    cout<<"Nilai yang ditunjuk A : "<<*A<<endl;
    cout<<"Alamat yang ditunjuk A : "<<A<<endl;
    cout<<"Nilai q : "<<q<<endl;
    cout<<"Nilai yang ditunjuk B : "<<*B<<endl;
    cout<<"Alamat yang ditunjuk B : "<<B<<endl;
    getch();
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.

```
Nilai p : 100
Nilai yang ditunjuk A : 100
Alamat yang ditunjuk A : 0x6ffdfc
Nilai q : 50
Nilai yang ditunjuk B : 50
Alamat yang ditunjuk B : 0x6ffdf8
Nilai p : 100
Nilai yang ditunjuk A : 100
Alamat yang ditunjuk A : 0x6ffdfc
Nilai q : 100
Nilai yang ditunjuk B : 100
Alamat yang ditunjuk B : 0x6ffdf8
```

Gambar 1. 11 Output Kode Program

6. Gantilah baris `*B = *A;` menjadi `B = A;`
7. Amati hasilnya
8. Bandingkan dengan sebelumnya.

1.8. Sorting

Sorting adalah proses pengurutan data yang telah disusun sebelumnya secara acak sehingga tersusun secara teratur menurut aturan-aturan tertentu, sortir bisa naik atau turun.

Sorting Algorithms adalah metode untuk mengatur kembali sejumlah besar item ke dalam beberapa urutan tertentu seperti tertinggi ke terendah, atau sebaliknya, atau bahkan dalam beberapa urutan abjad.

Algoritme ini mengambil daftar input, memprosesnya (yaitu, melakukan beberapa operasi di dalamnya) dan menghasilkan daftar yang diurutkan.

Contoh paling umum yang kita alami setiap hari adalah menyortir pakaian atau barang lain di situs web e-commerce baik berdasarkan harga terendah hingga tertinggi, atau daftar berdasarkan popularitas, atau urutan lainnya.

A. Selection Sort

Selection sort adalah algoritma pengurutan berbasis perbandingan sederhana. Itu ada di tempat dan tidak memerlukan memori

tambahan. Ide di balik algoritma ini cukup sederhana. Kami membagi array menjadi dua bagian: diurutkan dan tidak disortir. Bagian kiri adalah subarray yang diurutkan dan bagian kanan adalah subarray yang tidak disortir. Awalnya, subarray yang diurutkan kosong dan array yang tidak disortir adalah array yang diberikan secara lengkap. *Selection sort* membandingkan item saat ini dengan item berikutnya hingga item terakhir. Jika item lain ditemukan lebih kecil dari item saat ini, catat posisinya dan tukar.

B. Insertion Sort

Insertion sort adalah mekanisme pengurutan di mana array yang diurutkan dibangun dengan satu item pada satu waktu. Elemen-elemen array dibandingkan satu sama lain secara berurutan dan kemudian diatur secara bersamaan dalam beberapa urutan tertentu. Analogi tersebut dapat dipahami dari gaya kita menyusun setumpuk kartu. Jenis ini bekerja berdasarkan prinsip menyisipkan elemen pada posisi tertentu, oleh karena itu dinamakan *insertion sort*. Klasifikasi dilakukan dengan membandingkan data ke-*i* (dimana *i* dari yang kedua sampai yang terakhir) dengan data berikut. Jika ditemukan data yang lebih kecil, masukkan data ke depan sesuai dengan tempatnya.

1.9. Praktik Sorting

1. Cobalah program berikut, jalankan dan analisis hasilnya

```
//-----  
#include <iostream>  
#include "conio.h"  
using namespace std;  
//-----  
void baca_data(int A[], int n)  
{  
    int i;  
    for (i=0;i<n;i++)
```

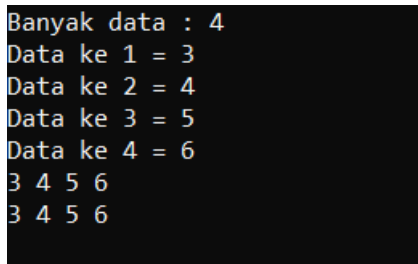
```

{
    cout<<"Data ke "<<(i+1)<<" = " ;
    cin>>A[i];
}
}
void cetak_data(const int A[], int n)
{
    int i;
    for (i=0; i<n;i++)
        cout<<A[i]<<" ";
    cout<<endl;
}
void insertion_sort(int A[], int n)
{
    int k,j,temp;
    for (k=0;k<n;k++)
    {
        temp=A[k];
        j=k-1;
        while ((temp<=A[j]) && (j>0))
        {
            A[j+1] = A[j];
            j=j-1;
        }
        if (temp>=A[j]) A[j+1]=temp;
        else
        {
            A[j+1]=A[j];
            A[j]=temp;
        }
    }
}

```

```
}  
int main()  
{  
    int data[10],n;  
    cout<<"Banyak data : ";  
    cin>>n;  
    baca_data(data,n);  
    cetak_data(data,n);  
    insertion_sort(data,n);  
    cetak_data(data,n);  
    getch();  
}  
//-----
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Banyak data : 4  
Data ke 1 = 3  
Data ke 2 = 4  
Data ke 3 = 5  
Data ke 4 = 6  
3 4 5 6  
3 4 5 6
```

Gambar 1. 12 Output Kode Program Sorting

2. Cobalah dengan beberapa kemungkinan data.
3. Coba juga program berikut dan amati hasilnya.

```
#include <iostream>  
#include "conio.h"  
using namespace std;  
  
void baca_data(int A[], int n)
```

```
{
    int i;
    for (i=0;i<n;i++)
    {
        cout<<"Data ke "<<i+1<<" = " ;
        cin>>A[i];
    }
}

void cetak_data(const int A[], int n)
{
    int i;
    for (i=0; i<n;i++)
        cout<<A[i]<<" ";
    cout<<endl;
}

void tukar(int &a, int &b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}

void Selection_sort(int A[], int n)
{
    int i,t,min;
    for (i=0;i<n;i++)
    {
        min=i;
        for (t=i+1;t<n;t++)
```

```
    if (A[t]<A[min])
        min=t;
    if (min!=i)
        tukar(A[i],A[min]);
    }
}
int main()
{
    int data[10],n;
    cout<<"Banyak data : ";
    cin>>n;
    baca_data(data,n);
    cetak_data(data,n);
    Selection_sort(data,n);
    cetak_data(data,n);
    getch();
}
```

1.10. Soal Latihan

1. Apa operasi yang valid pada pointer?
2. Apa itu pointer konstan? Bedakan antara pointer konstan dan pointer ke sebuah konstanta?
3. Bagaimana menemukan jumlah elemen dalam array jika ukuran array tidak disediakan?
4. Bisakah ukuran operator digunakan untuk mengetahui ukuran array yang diteruskan ke suatu fungsi?

BAB II

MERGE SORT, REKURSIF & QUICK SORT

Capaian Pembelajaran

1. Memahami karakteristik algoritma *merge sort*
2. Menggunakan algoritma merge sort untuk mengurutkan data
3. Memahami karakteristik algoritma *quick sort*
4. Menggunakan algoritma *quick sort* untuk mengurutkan data

2.1. Teori Singkat Merge Sort

Merge sort adalah salah satu algoritma pengurutan yang paling efisien. Ia bekerja berdasarkan prinsip *divide and conquer*. *Merge sort* berulang kali memecah daftar menjadi beberapa sublist hingga setiap sublist terdiri dari satu elemen dan menggabungkan sublist tersebut dengan cara yang menghasilkan daftar yang diurutkan.

Algoritma *merge sort* dilengkapi dengan metode *divide* dan *conquer*, yaitu setiap bagian dibagi, kemudian diselesaikan, dan digabungkan kembali. Pertama-tama bagilah data menjadi dua bagian. Bagian pertama adalah setengah dari semua data (jika datanya genap) atau setengah dikurangi satu (jika datanya ganjil), dan kemudian setiap blok dibagi lagi hingga hanya berisi satu data per blok.

Pendekatan *merge sort top-down* adalah metodologi yang menggunakan mekanisme rekursi. Itu dimulai di Atas dan berlanjut ke bawah, dengan setiap giliran rekursif menanyakan pertanyaan yang sama seperti "Apa yang harus dilakukan untuk mengurutkan array?" dan memiliki jawabannya sebagai, "bagi array menjadi dua, buat panggilan rekursif, dan gabungkan hasilnya.", Sampai seseorang mencapai bagian bawah pohon array.

Setelah itu, jika data pertama lebih besar dari data perantara +1, gabungkan dan bandingkan lagi di blok yang sama, jika ya, data

perantara +1 dipindahkan sebagai data pertama, lalu data pertama dipindahkan ke data kedua Ke tengah + 1, dan seterusnya, sampai terbentuk blok yang lengkap. Jadi metode merge sort adalah metode yang membutuhkan fungsi rekursif untuk menyelesaikannya.

Dengan ini, deskripsi algoritma diekspresikan dalam 3 langkah menggunakan model *divide* dan *conquer*. Langkah-langkah kerja Mergesort dijelaskan di bawah ini.

- **Divide**

Bagilah elemen-elemen dari kumpulan data menjadi dua bagian.

- **Conquer**

Conquer Setiap bagian secara rekursif memanggil proses sortir secara rekursif.

- **Kombinasi**

Gabungkan dua bagian secara rekursif untuk mendapatkan kumpulan data berurutan

Proses rekursif berhenti ketika mencapai elemen dasar. Ini terjadi ketika hanya ada satu item yang tersisa di bagian yang akan dipesan. Jenis item yang tersisa berarti bahwa bagian tersebut telah diurutkan berdasarkan seri.

2.2. Praktik Merge Sort

Ketikan kode program berikut, jalankan dan analisis hasilnya

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

int a[50];
void merge(int,int,int);
void merge_sort(int low,int high) {
int mid;
if(low<high) {
```

```

mid=(low+high)/2;
merge_sort(low,mid);
merge_sort(mid+1,high);
merge(low,mid,high);
}
}
void merge(int low,int mid,int high) {
int h,i,j,b[50],k;
h=low;
i=low;
j=mid+1;
while((h<=mid)&&(j<=high)) {
if(a[h]<=a[j]) {
b[i]=a[h]; h++;
}
else {
b[i]=a[j]; j++;
} i++;
}
if(h>mid) {
for(k=j;k<=high;k++) {
b[i]=a[k]; i++;
}
}
else {
for(k=h;k<=mid;k++) {
b[i]=a[k]; i++;
}
}
for(k=low;k<=high;k++)
a[k]=b[k];

```

```
    }
    int main() {
        int num,i;
        cout<<"*****"
        *****"<<endl;
        cout<<" MERGE SORT PROGRAM "<<endl;
        cout<<"*****"
        *****"<<endl;
        cout<<endl<<endl;
        cout<<"Masukkan Banyak Bilangan: ";cin>>num;
        cout<<endl;
        cout<<"Sekarang masukkan "<< num <<" Bilangan yang ingin
        Diurutkan:"<<endl;
        for(i=1;i<=num;i++) {
            cout<<"Bilangan ke-"<<i<<" ";cin>>a[i] ;
        }
        merge_sort(1,num);
        cout<<endl;
        cout<<"Hasil akhir pengurutan :"<<endl;
        cout<<endl;
        for(i=1;i<=num;i++)
            cout<<a[i]<<" ";
        cout<<endl<<endl<<endl<<endl;
        getch();
    }
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.

```
*****
MERGE SORT PROGRAM
*****

Masukkan Banyak Bilangan:
5

Sekarang masukkan 5 Bilangan yang ingin Diurutkan:
Bilangan ke-1 4
Bilangan ke-2 3
Bilangan ke-3 6
Bilangan ke-4 8
Bilangan ke-5 3

Hasil akhir pengurutan :
3 3 4 6 8
```

Gambar 2. 1 Output Kode Program Merge Sort

2.3. Teori Singkat Rekursif

Rekursi adalah secara langsung atau tidak langsung memanggil fungsinya sendiri, dan proses pemanggilannya disebut rekursi.

Masalah yang dapat diselesaikan secara rekursif adalah dengan membagi masalah menjadi satu atau lebih masalah kecil yang serupa.

Simple Cases adalah suatu kondisi yang dapat diselesaikan secara langsung tanpa rekursi, dan umumnya digunakan sebagai tanda berakhirnya rekursi.

Recursive Case Ini adalah kondisi yang diselesaikan dengan memanggil fungsi itu sendiri, dan semakin sedikit masalah yang mendekati *Recursive Case*.

2.4. Teori Singkat Quick Sort

Quick sort merupakan algoritma pembagi. Pertama, *quick sort* membagi daftar besar menjadi dua sub-daftar yang lebih kecil: elemen kecil dan elemen besar. *Quick sort* dapat mengurutkan subdaftar secara rekursif. Algoritma Quick Sort Metode quick sort dikembangkan oleh C. A. R Hoare pada tahun 1960, dan dimuat sebagai artikel di

“Computer Journal 5” pada April 1962. Algoritma sorting yang berdasarkan perbandingan dengan metoda *divide-and-conquer*. Disebut Quick Sort, karena Algoritma quick sort mengurutkan dengan sangat cepat, namun algoritma ini sangat kompleks dan diproses secara rekursif. Langkah-langkah prosesnya adalah:

1. Dapatkan elemen bernama *pivot* dalam daftar.
2. Susun ulang daftar sehingga elemen dengan nilai kurang dari *pivot* berada sebelum *pivot*, dan semua elemen dengan nilai lebih besar dari *pivot* berada setelah *pivot* (nilai yang sama setelah *pivot*). Setelah pemisah, *pivot* berada di posisi akhirnya dan operasi ini disebut *partition*.
3. Kemudian urutkan sublist item terkecil dan urutkan sublist item terbesar secara rekursif.
4. Kasus dasar rekursi adalah daftar nol atau satu, tidak diperlukan *sorting*.

Sebelum menggunakan algoritma apa pun, sangat penting bagi kita untuk memahami apa aplikasi dunia nyatanya. *Quick sort* menyediakan pendekatan cepat dan metode untuk menyortir daftar hal apa pun. Berikut ini adalah beberapa aplikasi di mana *Quick sort* digunakan.

1. Komputasi komersial: Digunakan di berbagai organisasi pemerintah dan swasta untuk tujuan menyortir berbagai data seperti menyortir akun/profil berdasarkan nama atau ID yang diberikan, menyortir transaksi berdasarkan waktu atau lokasi, menyortir file berdasarkan nama atau tanggal pembuatan, dan lain-lain.
2. Komputasi numerik: Sebagian besar algoritma yang dikembangkan secara efisien menggunakan antrian prioritas dan penyortiran balik untuk mencapai akurasi dalam semua perhitungan.
3. Pencarian informasi: Algoritma pengurutan membantu pencarian informasi yang lebih baik dan cara apa yang lebih cepat daripada mencapai pengurutan menggunakan pengurutan cepat.

Pada dasarnya, quick sort digunakan di mana-mana untuk hasil yang lebih cepat dan dalam kasus di mana ada batasan ruang.

2.5. Bubble Sort

Bubble sort, juga disebut sebagai perbandingan, adalah algoritma pengurutan sederhana yang berulang kali melewati daftar, membandingkan elemen yang berdekatan dan menukarnya jika urutannya salah. Ini adalah algoritma yang paling sederhana dan tidak efisien pada saat bersamaan. Namun, sangat perlu untuk mempelajarinya karena ini mewakili dasar-dasar pemilahan.

Bubble sort terutama digunakan dalam tujuan pendidikan untuk membantu siswa memahami dasar-dasar penyortiran.

Ini digunakan untuk mengidentifikasi apakah daftar sudah diurutkan. Ketika daftar sudah diurutkan (yang merupakan skenario kasus terbaik), kompleksitas *bubble sort* hanya $O(n)$.

Dalam kehidupan nyata, bubble sort dapat divisualisasikan ketika orang-orang dalam antrian yang ingin berdiri dalam urutan ketinggian yang bijaksana bertukar posisi di antara mereka sendiri sampai semua orang berdiri berdasarkan urutan ketinggian yang meningkat.

2.6. Praktik Rekursif & Quick Sort

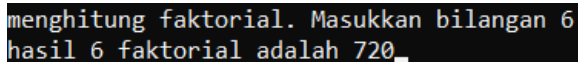
Tuliskan kode program dibawah ini, kemudian jalankan dan analisis hasilnya.

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

long int faktorial (int A);
int main()
/*fungsi ini tidak memberikan nilai balik. fungsi ini sebagai fungsi
utama*/
{
    int r,hasil;
    cout<<"menghitung faktorial. Masukkan bilangan ";
```

```
cin>>r;
hasil=faktorial(r);
cout<<"hasil "<<r<<" faktorial adalah "<<hasil;
getch();
}
long int faktorial(int A)
/*fungsi ini memberikan nilai balik berupa hasil faktorial dari data
inputan*/
{
    if (A==1)
        return(A);
    else
        return (A*faktorial(A-1));
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



menghitung faktorial. Masukkan bilangan 6
hasil 6 faktorial adalah 720_

Gambar 2. 2 Output Kode Program Quick Sort

Kemudian tambahkan lagi kode program dibawah ini

```
#include <iostream>
#include "conio.h"
#define max 20
using namespace std;

void quick_sort(int darr[max], int lb, int ub)
{
    int a;
    int up,down;
    int temp;
```

```
if (lb>=ub)
return;
a=darr[lb];
up=ub;
down=lb;

while (down < up)
{
while (darr[down] <= a)
down++;
while (darr[up]>a)
up--;
if(down<up)
{
temp=darr[down];
darr[down]=darr[up];
darr[up]=temp;
}
}
darr[lb]=darr[up];
darr[up]=a;

quick_sort(darr,lb,up-1);
quick_sort(darr,up+1,ub);
}

int main()
{
int arr[max];
int i,n,lb,ub;
lb=0;
```

```
cout<<"Masukkan banyak data yang ingin diurut: ";
cin>>n;

ub=n;
cout<<"Masukkan data-datanya: \n\n";
for(i=1;i<=n;i++)
{
    cout<<"\tdata ke- "<<i<<" : "; cin>>arr[i];
}
quick_sort(arr,lb,ub);
cout<<"\nHasil pengurutan data: ";
for(i=0; i<n;i++)
    cout<<" "<<arr[i];

cout<<"\n\nTekan sembarang tombol untuk keluar ";
getch();
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.

```
Masukkan banyak data yang ingin diurut: 6
Masukkan data-datanya:

    data ke- 1 : 4
    data ke- 2 : 3
    data ke- 3 : 6
    data ke- 4 : 4
    data ke- 5 : 2
    data ke- 6 : 9

Hasil pengurutan data: 2 3 4 4 6 8

Tekan sembarang tombol untuk keluar _
```

Gambar 2. 3 Output Kode Program Quick Sort

BAB III

SEARCHING & STACK

Capaian Pembelajaran

1. Memahami karakteristik algoritma searching
2. Menggunakan algoritma searching untuk pencarian data
3. Memahami konsep *stack* (tumpukan)
4. Mampu membuat program yang mengimplementasikan konsep tumpukan dengan larik dan link list

3.1. Teori Singkat Searching

Searching adalah metode untuk menemukan informasi dalam aplikasi menggunakan kunci. Ketika lokasi yang tepat dari informasi spesifik sebelumnya tidak diketahui, pencarian diperlukan untuk menemukan informasi spesifik dalam tabel. Pencarian selalu direpresentasikan dengan mengacu pada keberadaan sekumpulan data yang disimpan secara terorganisir, yang disebut table. Algoritma Pencarian *searching* untuk memeriksa elemen atau mengambil elemen dari struktur data mana pun yang menyimpannya. Berdasarkan jenis operasi pencarian, algoritma ini umumnya diklasifikasikan menjadi dua kategori. Adapun ketegori yang digunakan dalam metode pencarian, yaitu: sekuensial (*sequential search*) dan Pencarian biner (*binary search*).

A. Pencarian sekuensial (*sequential search*)

Pencarian sekuensial (*sequential search*), atau biasa disebut pencarian linier, menggunakan prinsip sebagai berikut: membandingkan data yang ada dengan konten yang dicari satu per satu secara berurutan. Pada dasarnya pencarian ini hanya mengulang dari 1 sampai jumlah data. Dalam setiap iterasi, data kei dibandingkan dengan

konten yang dicari. Jika sama, data telah ditemukan. Sebaliknya, jika tidak ada yang sama sampai akhir pengulangan, berarti data tersebut tidak ada.

B. Pencarian Biner (*Binary Search*)

Salah satu syarat untuk melakukan pencarian biner (*binary search*) adalah data sudah dalam keadaan terurut. Dengan kata lain, jika data tidak dalam keadaan terurut, pencarian biner tidak dapat dilakukan. Dalam kehidupan sehari-hari, sebenarnya kita sering menggunakan pencarian biner. Misalnya, ketika kita ingin mencari sebuah kata dalam kamus. Langkah-langkah untuk pencarian biner adalah:

1. Awalnya diambil dari posisi awal = 1 dan posisi akhir = n
2. Kemudian menggunakan rumus mean posisi = (posisi awal + posisi akhir) div 2 untuk mencari posisi data rata-rata.
3. Kemudian bandingkan data yang dicari dengan data tengah.
 - a. Jika sama, data ditemukan, Proses selesai
 - b. Jika lebih kecil, proses dilakukan kembali tetapi posisi akhir dianggap sama dengan posisi tengah -1,
 - c. Jika lebih besar, proses dilakukan kembali tetapi posisi awal dianggap sama dengan posisi tengah +1.
4. Ulangi langkah kedua hingga data ditemukan, atau tidak ditemukan.

Jika posisi awal data yang ditemukan lebih besar dari posisi akhir, pencarian biner akan berakhir. Jika posisi awal sudah lebih besar dari posisi akhir, berarti tidak ada data yang ditemukan.

3.2. Praktik Searching

Ketikan kode program dibawah ini, jalankan dan analisis hasilnya

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

```
void main()
{
    int i;
    int cari,ketemu;
    int A[100] ;

    cout<<"PROGRAM SEARCHING Linier\n";
    cout<<"masukkan 7 buah data : \n\n";
    for (i=1;i<=7;i++)
    {
        cout<<"masukkan data ke-<<i<<" = ";
        cin>>A[i] ;
    }

    cout<<endl;
    cout<<"Input bilangan yang dicari : ";
    cin>>cari;

    ketemu=0;
    for(i=0;i<=7;i++)
    {
        if (A[i]==cari)
        {
            ketemu=1;
            cout<<"Data ditemukan pada indeks ke-<<i;
        }
    }

    if (ketemu==0){
        cout<<"Data tidak ditemukan";
```

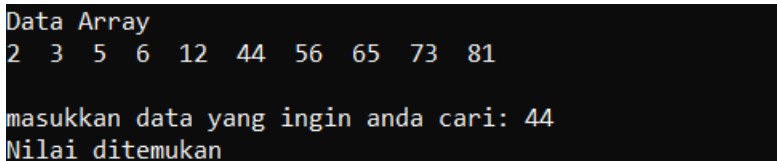
```
}  
  
getch();  
}
```

Kemudian tambahkan kode program dibawah ini

```
#include <iostream>  
#include <conio.h>  
using namespace std;  
  
int binary_s(int array[], int size, int elemen);  
int main()  
{  
    int size=10;  
    int data[10]={2, 3, 5, 6, 12, 44, 56, 65, 73 ,81} ;  
    cout<<"Data Array"<<endl;  
    int i;  
    for(i=0;i<size;i++)  
        cout<<data[i]<<" ";  
    cout<<endl<<endl<<"masukkan data yang ingin anda cari: ";  
    int cari;  
    cin>>cari;  
    // pencarian  
    int hasil;  
    hasil = binary_s(data, size, cari);  
    if (hasil==0)  
        cout<<"Nilai tidak ditemukan";  
    else  
        cout<<"Nilai ditemukan";  
    getch();  
}
```

```
int binary_s(int array[], int size, int elemen)
{
    int awal = 0;
    int akhir = size-1;
    int nilaiTengah = (awal+akhir)/2;
    while (nilaiTengah<=size && awal<=akhir)
    {
        nilaiTengah = (awal+akhir)/2;
        if (array[nilaiTengah]==elemen)
            return 1;
        else if (elemen<array[nilaiTengah])
            akhir = nilaiTengah-1;
        else
            awal = nilaiTengah+1;
    }
    return 0;
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Data Array
2 3 5 6 12 44 56 65 73 81
masukkan data yang ingin anda cari: 44
Nilai ditemukan
```

Gambar 3. 1 Output Program Searching

Adapun hasil ketika data yang diinputkan tidak ditemukan seperti gambar dibawah ini

```
Data Array
2 3 5 6 12 44 56 65 73 81

masukkan data yang ingin anda cari: 4
Nilai tidak ditemukan_
```

Gambar 3. 2 Output Program Searching

3.3. Teori Singkat Stack

Secara sederhana, *stack* (tumpukan) dapat diartikan sebagai kumpulan data, seolah-olah satu data ditempatkan di atas yang lain. Menambah dan menghapus data hanya dapat dilakukan dari ujung yang sama, yang disebut bagian atas tumpukan (*top of stack*). Pada kondisi tersebut, prinsip yang digunakan pada *stack* (tumpukan) adalah LIFO (*last in first out*).

Stack adalah tipe data abstrak dengan kapasitas terbatas (sudah ditentukan sebelumnya). Ini adalah struktur data sederhana yang memungkinkan penambahan dan penghapusan elemen dalam urutan tertentu. Setiap kali sebuah elemen ditambahkan, ia berada di atas tumpukan dan satu-satunya elemen yang dapat dihapus adalah elemen yang berada di atas tumpukan, seperti tumpukan objek.

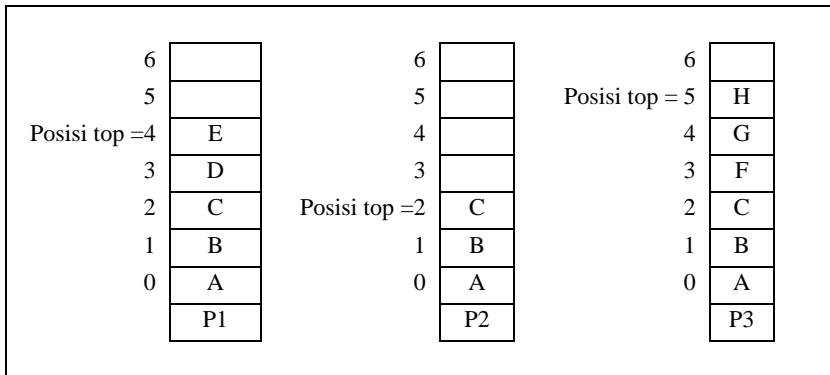
Dua operasi dasar dapat dilakukan pada *stack*, yaitu, operasi tambah data (PUSH) dan operasi pengurangan / penarikan data (POP). Tabel 3.1 mengilustrasikan operasi PUSH dan POP pada *stack*, yang diimplementasikan menggunakan matriks proses sekuensial ukuran 6, seperti yang ditunjukkan di bawah ini.

P1 : penambahan 3 data, yaitu A – B – C – D - E

P2 : pengurangan 2 data

P3 : penambahan 3 data, yaitu F – G – H

Tabel 3.1 Ilustrasi operasi PUSH dan POP pada Stack



Dalam kehidupan sehari-hari, realisasi *stack* dapat digambarkan dengan tumpukan kardus atau surat. Jika ada kardus atau surat baru maka kardus atau surat tersebut akan diletakkan di tumpukan paling atas, jika ingin mengambil kardus atau surat, maka kardus atau surat yang berada di posisi atas akan diambil terlebih dahulu.

3.4. Praktik Stack

Ketikkan kode program dibawah ini dengan contoh *stack* yang diimplementasikan dengan array

```
#include <iostream>
#include <conio.h>
using namespace std;
typedef int Item_type;
struct stack_tag
{
    int top;
    Item_type entry[10];
} Stack_type;

void Create()
```

```
{
    Stack_type.top=0;
}

bool Empty()
{
    return Stack_type.top<0;
}

bool Full()
{
    return Stack_type.top==10-1;
}

int Push(Item_type item)
{
    if (Full()==false)
    {
        Stack_type.top++;
        Stack_type.entry[Stack_type.top]=item;
    }
}

int Pop(Item_type &itemPop)
{
    if (Empty()==false)
    {
        itemPop = Stack_type.entry[Stack_type.top];
        Stack_type.top--;
    }
}
```

```
}

int Tampil()
{
    cout<<"isi tumpukan:\n";
    for (int i=1; i<=Stack_type.top;i++)
        cout<<Stack_type.entry[i]<<endl;
}

int main(){
int data;
Create();
cout<<"\nInput Data = ";cin>>data;
Push(data);
cout<<"\nInput Data = ";cin>>data;
Push(data);
cout<<"\n";
Tampil();
Pop(data);
cout<<"\n";
Tampil();
getch();
}
```

3.5. Soal Latihan

1. Apa itu *stack data structure*?
2. Bagaimana cara menerapkan *stack* menggunakan *queue*?

BAB IV

QUEUE & LINK LIST

Capaian Pembelajaran

1. Memahami konsep *queue* (antrian).
2. Mampu menulis program yang mengimplementasikan konsep array *queue* dan *link list*.
3. Mampu menulis program untuk mengimplementasikan operasi penambahan node sebelum dan di belakang *single list*.
4. Mampu membuat program untuk menambahkan node di tengah dan menghapus *single list*.

4.1. Teori Singkat Queue

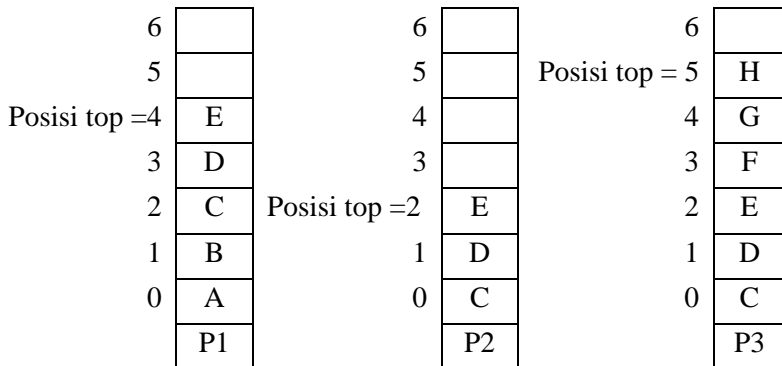
Queue (antrian) adalah sekumpulan data, penambahan elemen hanya dapat dilakukan pada salah satu ujung (disebut belakang atau *rear*), dan penghapusan atau pengambilan elemen dilakukan melalui ujung yang lain (disebut sisi depan atau *front*). *Queue* merupakan struktur data abstrak yang mirip dengan tumpukan. Tidak seperti tumpukan, antrian terbuka di kedua ujungnya. Salah satu ujungnya selalu digunakan untuk memasukkan data (*enqueue*) dan ujung lainnya untuk menghapus data (*dequeue*) Dalam antrian, prinsip yang dianut adalah FIFO (first in first out). Dua operasi dasar dapat dilakukan pada antrian, yaitu operasi penambahan data (*ENQUEUE*) dan operasi pengurangan/pemulihan data (*DEQUEUE*). Tabel 4.1 mengilustrasikan operasi *ENQUEUE* dan *DEQUEUE* pada antrian, yang diimplementasikan menggunakan array proses sekuensial berukuran 6, seperti yang ditunjukkan di bawah ini.

P1 : penambahan 3 data, yaitu A – B – C – D - E

P2 : pengurangan 2 data

P3 : penambahan 3 data, yaitu F – G – H

Tabel 4.1 Ilustrasi operasi *PUSH* dan *POP* pada *QUEUE*



Dalam kehidupan sehari-hari, antrian dapat digambarkan sebagai antrian pasien di poliklinik. Jika ada pasien baru maka pasien tersebut akan mendapatkan nomor antrian paling besar dan akan menjadi pasien terakhir yang mendapatkan pelayanan setelah pasien sebelumnya.

4.2. Praktik QUEUE

Ketikan kode program dibawah ini dengan contoh queue yang diimplementasikan dengan array.

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

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

Queue antrian;

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

```

```

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

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

void Enqueue(int data){
    if(IsEmpty()==1){
        antrian.head=antrian.tail=0;
        antrian.data[antrian.tail]=data;}
    else
        if(IsFull()==0){
            antrian.tail++;
            antrian.data[antrian.tail]=data;}}

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

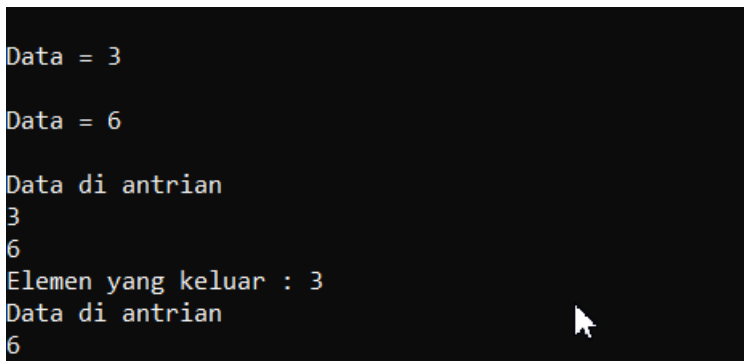
void Clear(){
    antrian.head=antrian.tail=-1;
    cout<<"data clear";}

```

```
void Tampil(){
    if(IsEmpty()==0){
        cout<<"\nData di antrian";
            for(int i=antrian.head;i<=antrian.tail;i++){
                cout<<"\n"<<antrian.data[i];}
        else cout<<"data kosong!\n";}

int main(){
int data;
Create();
cout<<"\nData = ";cin>>data;
Enqueue(data);
cout<<"\nData = ";cin>>data;
Enqueue(data);
Tampil();
cout<<"\nElemen yang keluar : "<<<Dequeue();
Tampil();
getch();
}
```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.



```
Data = 3
Data = 6
Data di antrian
3
6
Elemen yang keluar : 3
Data di antrian
6
```

Gambar 4. 1 Output Kode Program Queue

4.3. Teori Singkat Link List 1

Linked list adalah cara untuk menyimpan data dalam suatu struktur sehingga program dapat secara otomatis membuat lokasi baru untuk menyimpan data saat dibutuhkan. Secara rinci, sebuah program dapat menulis struktur atau definisi kelas yang berisi variabel yang berisi informasi yang terkandung di dalamnya, dan memiliki penunjuk ke struktur sesuai dengan tipe datanya.

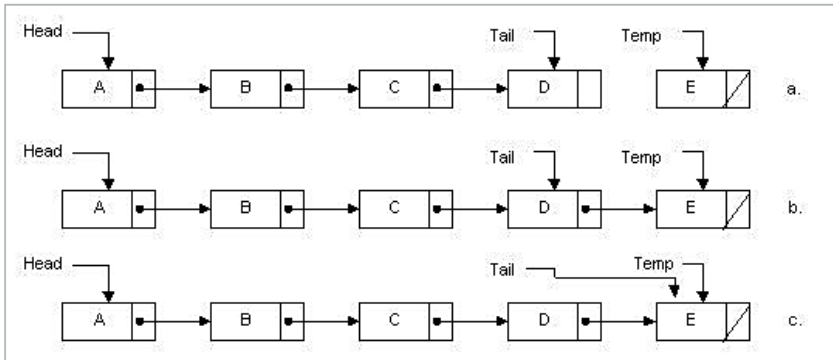
Struktur dinamis ini memiliki banyak keunggulan dibandingkan struktur array yang bersifat statis. Struktur ini lebih dinamis, karena banyak elemen dapat ditambahkan atau dikurangi dengan mudah dibandingkan dengan array dengan ukuran tetap. Selain itu, Anda dapat dengan mudah menyelesaikan operasi pada setiap elemen, seperti menyisipkan, menghapus, dan menambahkan.

Definisi link list dapat dituliskan secara sederhana dengan struktur berikut;

```
Struct node {  
    Int info;  
    Struct node *nextPtr;  
};
```

Struktur ini memiliki dua anggota, informasi integer dan pointer *nextPtr*. Variabel *pointer nextPtr* ini menunjuk ke struktur tipe struct node, sama dengan deklarasi utama.

Operasi yang dapat dilakukan pada *link list* adalah menambah dan menghapus elemen dalam daftar *link list*. Menambah dan menghapus dapat dilakukan di depan, belakang atau beberapa posisi dari *link list*.



Gambar 4. 2 Merupakan ilustrasi penambahan data di belakang link list.

4.4. Praktik Link List Penambahan Simpul Di Depan Dan Di Belakang

Ketikan kode program dibawah ini dengan contoh queue yang diimplementasikan dengan array.

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

typedef char Item_type;
typedef struct LinkList
{
    Item_type entry;
    struct LinkList *next;
} ListNode;

typedef struct list
{
    ListNode *awal;
    ListNode *akhir;
```

```

} List;

void Create(List *q)
{
    q->awal=q->akhir=NULL;
}

void Tambah_Belakang(List *q, Item_type item)
{
    ListNode *baru= new ListNode;
    baru->entry=item;
    if (q->awal==NULL)
        q->awal=baru;
    else
        q->akhir->next = baru;
    q->akhir=baru;
    q->akhir->next=NULL;
}

void Baca_Maju(List *q)
{
    ListNode *bantu=new ListNode;
    bantu=q->awal;
    while (bantu!=NULL)
    {
        cout<<bantu->entry;
        bantu=bantu->next;
        cout<<"-->";
    }
    cout<<"NULL";
    cout<<endl;
}

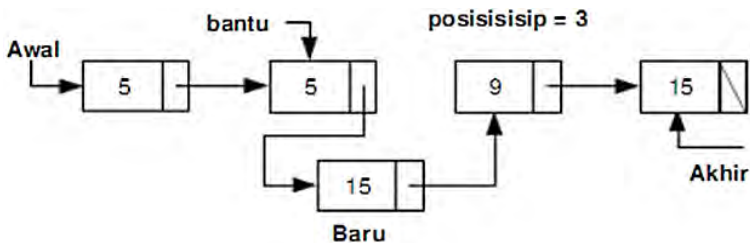
```

```
}  
void Tambah_Depan(List *q, Item_type item)  
{  
    ListNode *baru=new ListNode;  
    baru->entry=item;  
    if (q->awal==NULL)  
    {  
        q->akhir=baru;  
        baru->next = NULL;  
    }  
    else  
        baru->next = q->awal;  
    q->awal=baru;  
}  
void main()  
{  
    List *SingleList=new List;  
    char pilih, hasil, data,cari;  
    int tempat;  
    bool ada;  
    Create(SingleList);  
    Tambah_Belakang(SingleList, 'S');  
    Baca_Maju(SingleList);  
    Tambah_Belakang(SingleList, 'A');  
    Baca_Maju(SingleList);  
    Tambah_Depan(SingleList, 'T');  
    Baca_Maju(SingleList);  
    cout<<endl<<endl;  
    delete (SingleList);  
    getch();  
}
```

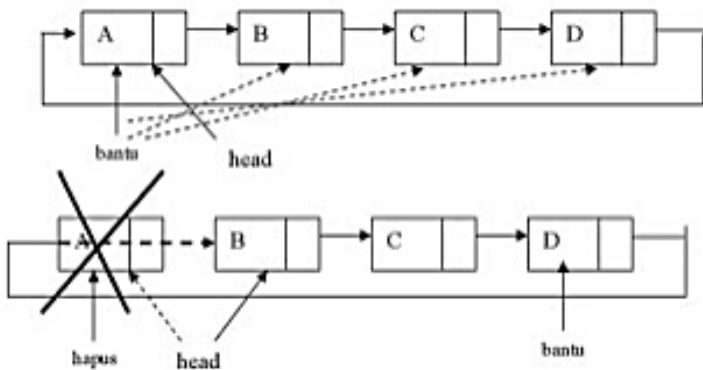
4.5. Teori Singkat Link List 2

Operasi yang dapat dilakukan pada daftar tertaut adalah menambah dan menghapus elemen dalam *linked list*. Menambah dan menghapus dapat dilakukan di depan, belakang atau beberapa posisi dari *linked list*.

Gambar 4.2 merupakan ilustrasi penambahan di posisi tertentu pada link list dan Gambar 4.3 merupakan ilustrasi penghapusan data pada link list.



Gambar 4. 3 Ilustrasi penambahan data di posisi tertentu



Gambar 4. 4 Ilustrasi penghapusan data di posisi depan

4.6. Praktik Link List Menambah Data Di Posisi Tertentu Dan Menghapus Data

Berikut kode program implementasi linklist untuk menambah data di posisi tertentu dan menghapus data.

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

typedef char Item_type;
typedef struct LinkList
{
    Item_type entry;
    struct LinkList *next;
} ListNode;

typedef struct list
{
    ListNode *awal;
    ListNode *akhir;
} List;

void Create(List *q)
{
    q->awal=q->akhir=NULL;
}

void Tambah_Belakang(List *q, Item_type item)
{
    ListNode *baru= new ListNode;
    baru->entry=item;
    if (q->awal==NULL)
        q->awal=baru;
    else
        q->akhir->next = baru;
```

```
q->akhir=baru;
q->akhir->next=NULL;
}

void Baca_Maju(List *q)
{
    ListNode *bantu=new ListNode;
    bantu=q->awal;
    while (bantu!=NULL)
    {
        cout<<bantu->entry;
        bantu=bantu->next;
        cout<<"-->";
    }
    cout<<"NULL";
    cout<<endl;
}

void Tambah_Depan(List *q, Item_type item)
{
    ListNode *baru=new ListNode;
    baru->entry=item;
    if (q->awal==NULL)
    {
        q->akhir=baru;
        baru->next = NULL;
    }
    else
        baru->next = q->awal;
    q->awal=baru;
}
```

```
void Tambah_Tengah(List *q, Item_type item, int Posisi)
{
    ListNode *baru=new ListNode;
    ListNode *bantu=new ListNode;
    baru->entry=item;
    if (q->awal==NULL)
    {
        q->awal=baru;
        q->akhir=baru;
        q->akhir->next=NULL;
    }
    else
    {
        if (Posisi==0)
        {
            baru->next = q->awal;
            q->awal = baru;
        }
        else
        {
            bantu=q->awal;
            for (int i=1;i<Posisi;i++)
            if (bantu->next!=NULL)
                bantu=bantu->next;
            baru->next=bantu->next;
            bantu->next=baru;
        }
    }
}
```

```

bool Cari_Data(Item_type cari, List *q)
{
    ListNode *bantu=new ListNode;
    bantu=q->awal;
    while (bantu!=NULL)
    {
        if (bantu->entry==cari)
            return 0;
        else
        {
            bantu=bantu->next;
            if (bantu==NULL)
                return 1;
        }
    }
}

void Hapus_Simpul(List *q, Item_type hapusan)
{
    ListNode *bantu=new ListNode;
    ListNode *hapus=new ListNode;
    if (q->awal==NULL)
        cout<<"List masih kosong"<<endl;
    else
        if (q->awal->entry==hapusan)
        {
            hapus=q->awal;
            q->awal=hapus->next;
            delete(hapus);
            cout<<"Ditemukan di awal list"<<endl;
        }
}

```

```
else
{
    bantu=q->awal;
    while ((bantu!=q->akhir)&& (bantu->next->entry!=hapus) )
        bantu=bantu->next;
    hapus=bantu->next;
    if (hapus==NULL)
        cout<<"Yes";
    else
        cout<<"No.";

    if (hapus!=NULL)
    {
        if (hapus!=q->akhir)
            bantu->next=hapus->next;
        else
        {
            q->akhir=bantu;
            q->akhir->next=NULL;
        }
        delete(hapus);
    }
    else
        cout<<"Simpul tidak ditemukan"<<endl;
}

int main()
{
    List *SingleList=new List;
    char pilih, hasil, data,cari;
    int tempat;
```

```

bool ada;
Create(SingleList);
Tambah_Belakang(SingleList, 'S');
Baca_Maju(SingleList);
Tambah_Belakang(SingleList, 'A');
Baca_Maju(SingleList);
Tambah_Depan(SingleList, 'T');
Baca_Maju(SingleList);
Tambah_Tengah(SingleList, 'P', 2);
Baca_Maju(SingleList);
cout<<endl<<endl;
Hapus_Simpul(SingleList, 'T');
Baca_Maju(SingleList);
delete(SingleList);
getch();
}

```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.

```

S-->NULL
S-->A-->NULL
I-->S-->A-->NULL
I-->S-->P-->A-->NULL

Ditemukan di awal list
S-->P-->A-->NULL
-----
Process exited after 5.447 seconds with return value 0
Press any key to continue . . .

```

Gambar 4. 5 Output kode program link list

4.7. Soal Latihan

1. Apa itu *linked list* dan apa saja tipe datanya?
2. Bagaimana cara menerapkan *queue* menggunakan *stack*?

BAB V

DOUBLE LINK LIST

Capaian Pembelajaran

1. Mampu menulis program untuk menambahkan node bolak-balik di node *double link list*.
2. Mampu menulis program untuk menambah dan menghapus simpul di tengah *double link list*.

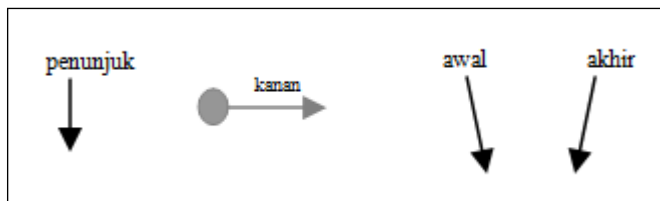
4.1 Teori Singkat Double Link list (Penambahan Simpul Didepan)

Double linked list hampir sama dengan *single linked list* yang dibahas pada bab sebelumnya, yaitu alokasi memori dinamis untuk menyimpan data. Perbedaannya adalah *double linked list* lebih fleksibel daripada *single linked list*, karena dalam *double linked list*, semua node memiliki 2 *pointer*, yang digunakan untuk menghubungkan *node* lain di kiri dan kanan.

Dalam konsep *double linked list*, terdapat 3 elemen pendukung penting, yaitu:

1. Penunjuk (biasa disebut *pointer*).

Pointer atau penunjuk yang dirujuk di sini adalah alat yang digunakan untuk menunjuk ke sebuah *node*, seperti pada Gambar 5.1.

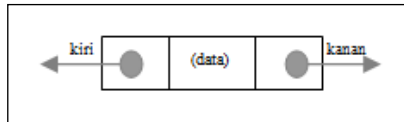


Gambar 5. 1 Ilustrasi Sebuah Penunjuk atau *Pointer*

Pointer dapat menunjuk ke sebuah node atau nilai null.

2. Simpul Ganda (biasa disebut *node*).

Node yang digunakan dalam bab ini adalah node ganda yang dijelaskan di bawah ini.



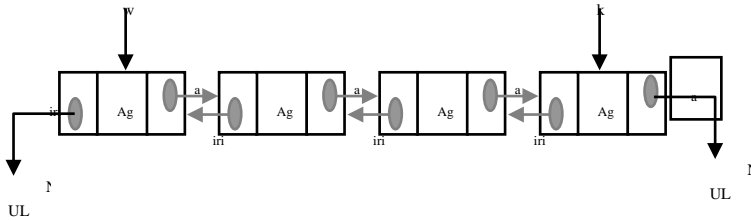
Gambar 5. 2 Ilustrasi Sebuah Simpul dalam *Double Linked List*

Simpul seperti ini disebut “simpul ganda” karena memiliki dua *pointer*, yaitu “kiri” dan “kanan”. Ini berarti bahwa *pointer* kanan menunjuk ke elemen di sebelah kanan dan penunjuk kiri menunjuk ke elemen di sebelah kiri. Gambar 10.2 mengilustrasikan sebuah node dalam daftar tertaut ganda. Dan (data) adalah data yang digunakan pada node, kiri adalah *pointer* ke node sebelumnya, dan kanan adalah *pointer* ke node berikutnya. Untuk membuat simpul ganda, dapat mendeklarasikannya dengan membuat struktur `strNode`, yang memiliki 2 anggota bertipe `strNode`, yaitu kanan dan kiri sebagai *pointer*, kemudian dari struktur `strNode` adalah `Node`, seperti yang ditunjukkan pada contoh berikut:

```
typedef struct strSimpul
{
    data masukkan;
    struct strSimpul *kanan;
    struct strSimpul *kiri;
} Simpul;
```

3. Senarai Berantai Ganda atau *Double Linked List* itu sendiri.

Double linked list adalah sekelompok node yang saling berhubungan. Satu-satunya perbedaan dalam daftar ini adalah bahwa setiap node memiliki 2 konektor, kiri dan kanan, seperti yang ditunjukkan pada gambar 5.3.



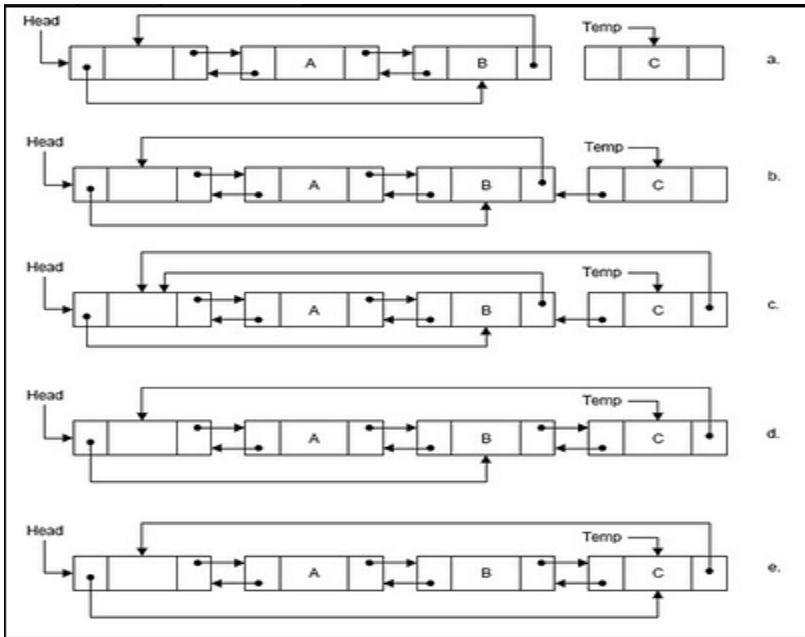
Gambar 5. 3 Ilustrasi Sebuah *Double Linked List*

Untuk membuat *double linked list*, dapat mendeklarasikannya dengan membuat struktur `strMultipleListen`, yang memiliki 2 anggota bertipe `Node`, yaitu awal dan akhir sebagai *pointer*. Jadi alias untuk struktur `strMultipleSeenrai` adalah `MultipleSeeList` juga diperlukan fungsi untuk membuat *double linked list*, nama dibuat, parameternya adalah `* q` bentuk *pointer*, dan nilai awal anggota awal dan akhir adalah kosong (null), seperti yang ditunjukkan pada contoh berikut ini;

```
typedef struct strSeneraiGanda
{
    Simpul *awal;
    Simpul *akhir;
} SeneraiGanda;

void buat(SeneraiGanda *q)
{
    q->awal=NULL;
    q->akhir=NULL;
}
```

Gambar 5.4 merupakan ilustrasi penambahan data di belakang pada *double link list*.



Gambar 5. 4 Ilustrasi penambahan data di belakang pada *double link list*

4.2 Praktik Penambahan Simpul Didepan

Berikut contoh implementasi Double Link List untuk menambah data didepan, dibelakang dan membaca data maju.

```
//-----
#pragma hdrstop
#include <iostream>
#include<stdio.h>
#include<conio.h>
#pragma argsused
using namespace std;
//-----
typedef char data;
typedef struct strSimpul
{
```

```
data masukkan;
struct strSimpul *kanan;
struct strSimpul *kiri;
} Simpul;

typedef struct strSeneraiGanda
{
    Simpul *awal;
    Simpul *akhir;
} SeneraiGanda;

void buat(SeneraiGanda *q)
{
    q->awal=NULL;
    q->akhir=NULL;
}

void Tambah_Depan(SeneraiGanda *q, data elemen)
{
    Simpul *baru=new Simpul();
    baru->masukkan=elemen;
    if (q->awal==NULL && q->akhir==NULL)
    {
        q->awal=baru;
        q->akhir=baru;
        baru->kanan=NULL;
        baru->kiri=NULL;
    }
    else
    {
        q->awal->kiri=baru;
    }
}
```



```

    baru->kanan=q->awal;
    q->awal=baru;
    q->awal->kiri=NULL;
}
}

void Tambah_Belakang(SeneraiGanda *q, data elemen)
{
    Simpul *baru=new Simpul();
    baru->masukkan=elemen;
    if (q->awal==NULL && q->akhir==NULL)
    {
        q->awal=baru;
        q->akhir=baru;
        baru->kanan=NULL;
        baru->kiri=NULL;
    }
    else
    {
        q->akhir->kanan=baru;
        baru->kiri=q->akhir;
        q->akhir=baru;
        q->akhir->kanan=NULL;
    }
}

void Baca_Maju(SeneraiGanda *q)
{
    Simpul *bantu=new Simpul();
    bantu=q->awal;
    while (bantu!=NULL)

```

```
{
    cout<<bantu->masukkan<<" ";
    bantu=bantu->kanan;
}
cout<<endl;
}

int main()
{
    SeneraiGanda *SnrGd=new SeneraiGanda();
    buat(SnrGd);
    data hasil;
    char pilih='1';
    while (pilih=='1' || pilih=='2' || pilih=='3')
    {
        cout<<"Pilih menu"<<endl;
        cout<<"1. Tambah Depan"<<endl;
        cout<<"2. Tambah Belakang"<<endl;
        cout<<"3. Baca Maju "<<endl;
        cout<<"4. Selesai "<<endl;
        cout<<"Pilihan = ";
        cin>>pilih;
        switch (pilih)
        {
            case '1' :{
                cout<<"Masukkan datanya : ";
                cin>>hasil;
                Tambah_Depan(SnrGd, hasil);
            }
            break;
            case '2' :{
```

```
        cout<<"Masukkan datanya : ";
        cin>>hasil;
        Tambah_Belakang(SnrGd, hasil);
    }
    break;
case '3' :{

    Baca_Maju(SnrGd);
    }
    break;
default : {
    cout<<"Selesai. Tekan enter";
    }
    break;
}
}
getch();
delete(SnrGd);
}
//-----
```

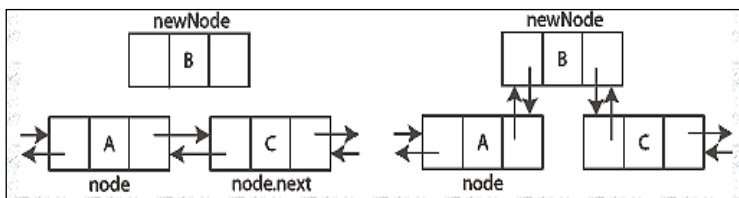
Hasil dari kode program diatas dapat dilihat seperti gambar dibawah ini.

```
Pilih menu
1. Tambah Depan
2. Tambah Belakang
3. Baca Maju
4. Selesai
Pilihan = 2
Masukkan datanya : 4
Pilih menu
1. Tambah Depan
2. Tambah Belakang
3. Baca Maju
4. Selesai
Pilihan = 4
Selesai. Tekan enter_
```

Gambar 5. 5 Output program penambahan simpul

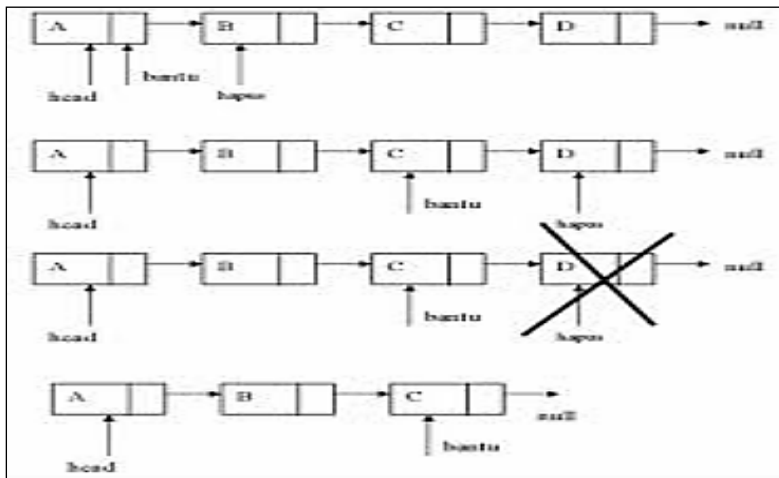
4.3 Teori Singkat Double Link list (Penambahan Simpul Ditengah)

Ada tiga jenis penambahan simpul pada *double link list*, yaitu menambahkan sebelum, sesudah (dibahas dalam 5.1) dan tengah, yang akan dibahas dalam buku teks ini. Gambar 5.5 menunjukkan instruksi untuk menambahkan data di tengah *double link list*.



Gambar 5. 6 Ilustrasi Penambahan Data di Tengah pada *Double Link List*

Ada juga tiga operasi untuk menghapus node dalam *double link list*, yaitu menghapus node depan, belakang, dan tengah. Namun dengan menggunakan *double link list*, proses pencarian node yang akan dihapus menjadi lebih cepat. Untuk menghapus node, diperlukan variabel *pointer* tambahan, yaitu variabel tambahan untuk menunjukkan node mana yang akan dihapus.



Gambar 5. 7 Ilustrasi Penghapusan Data pada *Double Link List*

4.4 Praktik Penambahan Simpul Ditengah

Berikut contoh implementasi *double link list* seperti pada point 5.2, ditambah dengan operasi menambah data di tengah, menghapus data dan membaca data mundur.

```
//Tuliskan program yang sudah dibuat pada point 5.2 (Praktik
Penambahan Simpul Didepan), sampai sebelum void main()
//Kemudian lanjutkan dengan program berikut

void Tambah_Tengah(SeneraiGanda *q, data elemen, int Posisi)
{
    Simpul *baru=new Simpul();
    Simpul *bantu=new Simpul();
    Simpul *bantu1=new Simpul();
    baru->masukkan=elemen;
    if (q->awal==NULL && q->akhir==NULL) //masih kosong
    {
        q->awal=baru;
        q->awal->kiri=NULL;
        q->akhir=baru;
    }
}
```

```

q->akhir->kanan=NULL;
}
else
{
if (Posisi==0) //posisi pertama, tambah depan
{
q->awal->kiri=baru;
baru->kanan=q->awal;
q->awal=baru;
q->awal->kiri=NULL;
}
else
{
bantu=q->awal;
for (int i=1;i<Posisi;i++)
if (bantu->kanan!=NULL)
    bantu=bantu->kanan;
if (bantu->kanan==NULL) //posisi terakhir, tambah belakang
{
    q->akhir->kanan=baru;
    baru->kiri=q->akhir;
    q->akhir=baru;
    q->akhir->kanan=NULL;}
else //posisi tengah beneran
{
    bantu1=bantu->kanan;
    baru->kiri=bantu;
    baru->kanan=bantu1;
    bantu->kanan=baru;
    bantu1->kiri=baru;}
}
}

```

```

}
}

void Hapus_Simpul(SeneraiGanda *q, data elemen)
{
    Simpul *bantu=new Simpul();
    Simpul *bantu1=new Simpul();
    Simpul *hapus=new Simpul();
    if (q->awal==NULL && q->akhir==NULL)
        cout<<"List masih kosong"<<endl;
    else
        if (q->awal->masukkan==elemen)//hapus awal list
            {
                hapus=q->awal;
                q->awal=hapus->kanan;
                q->awal->kiri=NULL;
                delete(hapus);
                cout<<"Ditemukan list paling kiri"<<endl;
            }
        else
            {
                bantu=q->awal;
                while ((bantu->kanan->kanan!=NULL)&&
                    (elemen!=bantu->kanan->masukkan))
                    bantu=bantu->kanan;
                if ((bantu->kanan->kanan==NULL)&&(elemen!=bantu->kanan->mas
                    ukkan))
                    cout<<"Data tidak ditemukan"<<endl;
                else
                    {hapus=bantu->kanan;

```

```
if (hapus!=NULL)
{
    if (hapus!=q->akhir)//hapus di tengah list
    {
        bantu1=hapus->kanan;
        bantu->kanan=hapus->kanan;
        bantu1->kiri=hapus->kiri;
        cout<<"Ditemukan di tengah list"<<endl;
    }
    else //hapus akhir list
    {
        q->akhir=bantu;
        q->akhir->kanan=NULL;
        cout<<"Ditemukan di akhir list"<<endl;
    }
    delete(hapus);
}
}
}
}

void Baca_Maju(SeneraiGanda *q)
{
    Simpul *bantu=new Simpul();
    bantu=q->awal;
    while (bantu!=NULL)
    {
        cout<<bantu->masukkan<<" ";
        bantu=bantu->kanan;
    }
    cout<<endl;
```



```

}

void Baca_Mundur(SeneraiGanda *q)
{
    Simpul *bantu=new Simpul();
    bantu=q->akhir;
    while (bantu!=NULL)
    {
        cout<<bantu->masukkan<<" ";
        bantu=bantu->kiri;
    }
    cout<<endl;
}

void main()
{
    SeneraiGanda *SnrGd=new SeneraiGanda();
    buat(SnrGd);
    int tempat;
    data hasil, cari;
    char pilih='1';
    while
(pilih=='1'||pilih=='2'||pilih=='3'||pilih=='4'||pilih=='5'||pilih=='6')
    {
        cout<<"Pilih menu"<<endl;
        cout<<"1. Tambah Depan"<<endl;
        cout<<"2. Tambah Belakang"<<endl;
        cout<<"3. Tambah Tengah"<<endl;
        cout<<"4. Hapus Data"<<endl;
        cout<<"5. Baca Maju"<<endl;
        cout<<"6. Baca Mundur "<<endl;
    }
}

```

```
cout<<"7. Selesai "<<endl;
cout<<"Pilihan = ";
cin>>pilih;
switch (pilih)
{
    case '1' :{
                cout<<"Masukkan datanya : ";
                cin>>hasil;
                Tambah_Depan(SnrGd, hasil);
                clrscr();
            }
        break;
    case '2' :{
                cout<<"Masukkan datanya : ";
                cin>>hasil;
                Tambah_Belakang(SnrGd, hasil);
                clrscr();
            }
        break;
    case '3' :{
                cout<<"Masukkan datanya : ";
                cin>>hasil;
                cout<<"Masukkan posisinya : ";
                cin>>tempat;
                Tambah_Tengah(SnrGd, hasil, tempat);
                clrscr();
            }
        break;
    case '4' :{
                clrscr();
                cout<<"Mau hapus simpul apa? ";
```

```
        cin>>cari;
        Hapus_Simpul(SnrGd, cari);
        clrscr();
    }
    break;
case '5' :{
            clrscr();
            Baca_Maju(SnrGd);
        }
    break;
case '6' :{
        clrscr();
        Baca_Mundur(SnrGd);}
    break;
default : {
            cout<<"Selesai. Tekan enter";
        }
    break;
}
}
getch();
delete(SnrGd);
}
//-----
```

4.5 Soal Latihan

- 1. Jelaskan apa yang dimaksud dengan double link list dan buatlah contoh program?
- 2. Bagaimana implementasi penambahan simpul ditengah?

BAB VI

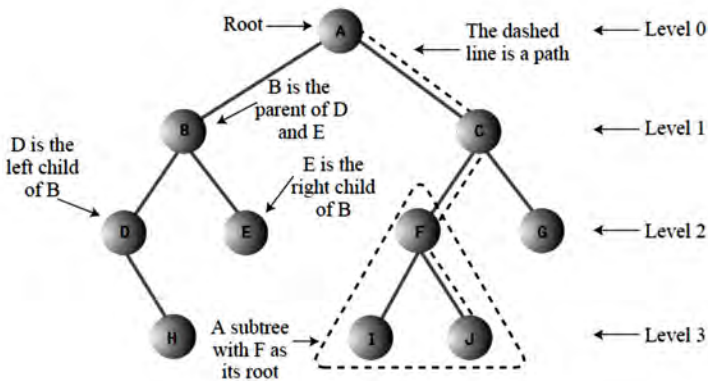
BINARY TREE

Capaian Pembelajaran

1. Mengetahui dan memahami tentang *binary tree*
2. Dapat membuat *binary tree*
3. Dapat melakukan penelusuran terhadap *binary tree*

6.1. Teori Singkat Binary Tree

Binary tree adalah *tree* syaratnya setiap node hanya boleh memiliki paling banyak dua *subtree*, dan kedua *subtree* tersebut harus dipisahkan. Dengan definisi ini, setiap node di *binary tree* hanya dapat memiliki paling banyak dua *child*.



H, E, I, J, and G are leaf nodes

Gambar 6.1 Binary Tree

Istilah-istilah umum dalam tree:

- a. Predecessor node yang berada di atas node tertentu.
- b. Successor node yang berada dibawah node tertentu.

c. Ancestor	seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama.
d. Descendant	seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama.
e. Parent	predesesor satu level di atas satu node.
f. Child	successor satu level di bawah suatu node.
g. Sibling	node-node yang memiliki parent yang sama dengan suatu node.
h. Subtree	bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
i. Size	banyaknya node dalam suatu tree.
j. Height	banyaknya tingkatan/level dalam suatu tree.
k. Root	satu-satunya node khusus dalam tree yang tak punya predesesor.
l. Leaf	Node-node dalam tree yang tak memiliki successor.
m. Degree	Banyaknya child yang dimiliki suatu node

6.2. Praktik Binary Tree

Tuliskan kode program dibawah ini sebagai contoh implementasi Binary Tree.

```
#include <iostream>
#include <cstdlib>
using namespace std;
class BinarySearchTree
{
private:
struct nodeTree
{
nodeTree* left;
nodeTree* right;
int data;

```

```

};
nodeTree* root;
public:
BinarySearchTree()
{
root = NULL;
}
bool isEmpty() const { return root==NULL; }
void print_inorder();
void inorder(nodeTree*);
void print_preorder();
void preorder(nodeTree*);
void print_postorder();
void postorder(nodeTree*);
void insert(int);
void remove(int);
};
void BinarySearchTree::insert(int d)
{
nodeTree* t = new nodeTree;
nodeTree* parent;
t->data = d;
t->left = NULL;
t->right = NULL;
parent = NULL;
if(isEmpty()) root = t;
else
{
nodeTree* current;
current = root;
while(current)
{
parent = current;
if(t->data > current->data) current = current->right;
else current = current->left;
}
if(t->data < parent->data)
parent->left = t;
else
parent->right = t;
}
}

```

```

}
}
void BinarySearchTree::remove(int d)
{
//Locate the element
bool found = false;
if(isEmpty())
{
cout<<" This Tree is empty! "<<endl;
return;
}
nodeTree* current;
nodeTree* parent;
current = root;
while(current != NULL)
{
if(current->data == d)
{
found = true;
break;
}
else
{
parent = current;
if(d>current->data) current = current->right;
else current = current->left;
}
}
if(!found)
{
cout<<" Data not found! "<<endl;
return;
}
// Node dengan single child
if((current->left == NULL && current->right != NULL)|| (current-
>left != NULL
&& current->right == NULL))
{
if(current->left == NULL && current->right != NULL)
{
if(parent->left == current)

```

```

{
parent->left = current->right;
delete current;
}
else
{
parent->right = current->right;
delete current;
}
}
else
{
if(parent->left == current)
{
parent->left = current->left;
delete current;
}
else
{
parent->right = current->left;
delete current;
}
}
return;
}
// node tidak mempunyai child node
if( current->left == NULL && current->right == NULL)
{
if(parent->left == current ) parent->left = NULL;
else parent->right = NULL;
delete current;
return;
}
//Node dengan 2 child node
// ganti node dengan nilai terkecil di subtree bagain kanan
if (current->left != NULL && current->right != NULL)
{
nodeTree* temp;
temp = current->right;
if((temp->left == NULL) && (temp->right == NULL))
{

```

```

current = temp;
delete temp;
current->right = NULL;
}
else
{
if((current->right)->left != NULL)
{
nodeTree* lcurrent;
nodeTree* lcurrp;
lcurrp = current->right;
lcurrent = (current->right)->left;
while(lcurrent->left != NULL)
{
lcurrp = lcurrent;
lcurrent = lcurrent->left;
}
current->data = lcurrent->data;
delete lcurrent;
lcurrp->left = NULL;
}
else
{
nodeTree* tmp2;
tmp2 = current->right;
current->data = tmp2->data;
current->right = tmp2->right;
delete tmp2;
}
}
return;
}
}

void BinarySearchTree::print_inorder()
{
inorder(root);
}
void BinarySearchTree::inorder(nodeTree* p)
{
if(p != NULL)

```

```

{
if(p->left) inorder(p->left);
cout<<" "<<p->data<<" ";
if(p->right) inorder(p->right);
}
else return;
}
void BinarySearchTree::print_preorder()
{
preorder(root);
}
void BinarySearchTree::preorder(nodeTree* p)
{
if(p != NULL)
{
cout<<" "<<p->data<<" ";
if(p->left) preorder(p->left);
if(p->right) preorder(p->right);
}
else return;
}
void BinarySearchTree::print_postorder()
{
postorder(root);
}
void BinarySearchTree::postorder(nodeTree* p)
{
if(p != NULL)
{
if(p->left) postorder(p->left);
if(p->right) postorder(p->right);
cout<<" "<<p->data<<" ";
}
else return;
}
int main()
{
BinarySearchTree b;
int ch,tmp,tmp1;
while(1)
{

```

```

cout<<endl<<endl;
cout<<" Binary Search Tree Operations "<<endl;
cout<<" ----- "<<endl;
cout<<" 1. Insertion/Creation "<<endl;
cout<<" 2. In-Order Traversal "<<endl;
cout<<" 3. Pre-Order Traversal "<<endl;
cout<<" 4. Post-Order Traversal "<<endl;
cout<<" 5. Removal "<<endl;
cout<<" 6. Exit "<<endl;
cout<<" Enter your choice : ";
cin>>ch;
switch(ch)
{
case 1 : cout<<" Enter Number to be inserted : ";
cin>>tmp;
b.insert(tmp);
break;
case 2 : cout<<endl;
cout<<" In-Order Traversal "<<endl;
cout<<" -----"<<endl;
b.print_inorder();
break;
case 3 : cout<<endl;
cout<<" Pre-Order Traversal "<<endl;
cout<<" -----"<<endl;
b.print_preorder();
break;
case 4 : cout<<endl;
cout<<" Post-Order Traversal "<<endl;
cout<<" -----"<<endl;
b.print_postorder();
break;
case 5 : cout<<" Enter data to be deleted : ";
cin>>tmp1;
b.remove(tmp1);
break;
case 6 :
return 0;
}
}
}
}

```

Hasil dari kode program diatas dapat dilihat seperti gambar dibawah.

```
Binary Search Tree Operations
-----
1. Insertion/Creation
2. In-Order Traversal
3. Pre-Order Traversal
4. Post-Order Traversal
5. Removal
6. Exit
Enter your choice : 2

In-Order Traversal
-----

Binary Search Tree Operations
-----
1. Insertion/Creation
2. In-Order Traversal
3. Pre-Order Traversal
4. Post-Order Traversal
5. Removal
6. Exit
Enter your choice : 3

Binary Search Tree Operations
-----
1. Insertion/Creation
2. In-Order Traversal
3. Pre-Order Traversal
4. Post-Order Traversal
5. Removal
6. Exit
Enter your choice : 4

Post-Order Traversal
-----

Binary Search Tree Operations
-----
1. Insertion/Creation
2. In-Order Traversal
3. Pre-Order Traversal
4. Post-Order Traversal
5. Removal
```

Gambar 6. 1 Output program Binary Tree

Selanjutnya tuliskan kode program dibawah ini sebagai contoh implementasi *Binary Tree* dan analisis hasilnya

```
//-----
#pragma hdrstop
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<stdlib>
#include<string>
#include<fstream>
//-----
#pragma argsused

typedef int Key_type;
typedef struct item_tag
{
    int key;
```

```

int no;
char *posisi;
}Item_type;
typedef struct node_tag
{
    Item_type info;
    struct node_tag *kanan;
    struct node_tag *kiri;
} Node_type;

int nourut=0;

void CreateTree(Node_type *root)
{
    root = NULL;
}

bool TreeEmpty(Node_type *root)
{
    return root==NULL;
}

Node_type *MakeNode(Item_type info)
{
    Node_type *p=NULL;
    if ((p=new (Node_type))==NULL)
        cout<<"Tidak ada tempat";
    p->info.key=info.key;
    p->kiri=p->kanan=NULL;
    return p;
}

```

```

Node_type *InsertTree(Node_type *root, Node_type *newnode)
{
    Node_type *p=root;
    while (p!=NULL)
    {
        if (newnode->info.key < p->info.key) //node baru lebih kecil dari
        p
        if (p->kiri) p=p->kiri;
        else
        {p->kiri = newnode;
        p->kiri->info.posisi=(char*)" kiri";
        p->kiri->info.no=++nourut;break;
        }
        else if (newnode->info.key > p->info.key)
        if (p->kanan) p=p->kanan;
        else
        {p->kanan=newnode;
        p->kanan->info.posisi=(char*)" kanan";
        p->kanan->info.no=++nourut;break;
        }
        else
        { cout<<"Kunci duplikat"<<endl;
        p=NULL;
        }
    } //akhir while
    newnode->kiri=newnode->kanan=NULL;
    if (root==NULL)
    {
        root=newnode;
        root->info.posisi=(char*)" root");
    }
}

```

```

    root->info.no=++nourut;
    }

    return root;
}

Node_type *TreeSearch(Node_type *p, Key_type target)
{
    if (p)
        if (target < p->info.key)
            p = TreeSearch(p->kiri,target);
        else if (target > p->info.key)
            p = TreeSearch(p->kanan, target);
    return p;
}

void main()
{
    Node_type *root=NULL;
    Node_type *p=root;
    Item_type info;
    //data ke 1
    info.key=10;
    p=root;
    Node_type* q=MakeNode(info);
    root=InsertTree(root,q);
    //data ke 2
    info.key=5;
    p=root;
    q=MakeNode(info);
    root=InsertTree(root,q);
}

```

```

//data ke 3
info.key=15;
p=root;
q=MakeNode(info);
root=InsertTree(root,q);

p=TreeSearch(p,info.key);
if (p)
{
    cout<<"Angka yang dicari "<<p->info.key<<endl;
    cout<<"Berada di posisi "<<p->info.posisi;
}
else cout<<"Tidak ada he..."<<endl;
delete(root);
getch();
}
//-----

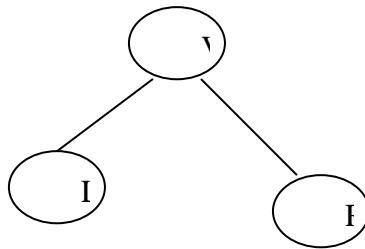
```

Jalankan dan amati hasilnya

6.3. Teori Singkat Binary Tree (Sesi 2)

Binary tree adalah pohon biner yang pada setiap node menyimpan data atau kunci yang lebih besar dari semua node pada sub tree kirinya yang lebih kecil dari semua node pada sub tree kanannya. Pada node tertentu, ada tiga kunjungan yang kita lakukan dengan aturan: kita mengunjungi node itu sendiri; kita melintasi subpohon kiri; kita melintasi subpohon kanan. Jika kita beri nama tiga kunjungan ini sebagai V, L, R, maka ada enam cara penelusuran:

VLR	LVR	LRV
VRL	RVL	RLV



Gambar 6.2. Pohon Biner yang mempunyai 2 Cabang.

Berdasarkan konvensi dasar, enam pilihan ini dikurangi menjadi tiga dengan memilih yang melintasi kiri sebelum kanan. Tiga lainnya sama. Tiga yang kita pilih kita beri nama:

VLR	LVR	LRV
Preorder	Inorder	Postorder

Kunjungan *inorder* adalah kunjungan dimana node yang dikunjungi mempunyai urutan dikunjungi node di sebelah kiri, kemudian mengunjungi node asli dan terakhir kunjungi node di sebelah kanan.

Kunjungan *preorde* adalah urutan kunjungan dari node mengunjungi node utama, kemudian mengunjungi node di sebelah kiri, dan terakhir mengunjungi node di sebelah kanan.

Kunjungan *postorder* adalah kunjungan dimana node yang berkunjung diperintahkan untuk mengunjungi node (L) di subtree kiri, kemudian mengunjungi node di sebelah kanan (R), dan terakhir mengunjungi node itu sendiri (V).

6.4. Praktik Binary Tree (Sesi 2)

Ketikkan kode program dibawah ini;

```
//tuliskan kembali program pada praktik Bab V (Point 5.4 Praktik
Penambahan Simpul Ditengah) sampai sebelum void main()
//kemudian lanjutkan dengan program berikut ini.

void Visit(Node_type *p)
```

```
{ cout<<"("<<p->info.no<<": "<<p->info.key<<");}
```

```
void Inorder(Node_type *root, char *s)
```

```
{
if (root)
{
    Inorder(root->kiri," Kiri");
    cout<<s<<" ";Visit(root);
    Inorder(root->kanan," Kanan");
} else cout<<"("<<s<<" NULL)"<<endl;
}
```

```
void Preorder(Node_type *root, char *s)
```

```
{
if (root)
{
    cout<<s<<" ";Visit(root);
    Preorder(root->kiri, " Kiri");
    Preorder(root->kanan, " Kanan");
} else cout<<"("<<s<<" NULL)"<<endl;
}
```

```
void Postorder(Node_type *root, char *s)
```

```
{
if (root)
{
    Postorder(root->kiri," Kiri");
    Postorder(root->kanan," Kanan");
    cout<<s<<" ";Visit(root);
} else cout<<"("<<s<<" NULL)"<<endl;
}
```

```
Node_type *TreeSearch(Node_type *p, Key_type target)
```

```

{
  if (p)
    if (target < p->info.key)
      p = TreeSearch(p->kiri,target);
    else if (target > p->info.key)
      p = TreeSearch(p->kanan, target);
    return p;
}

void main()
{
  //tuliskan disini koding untuk membuat binary tree
  //kemudian lanjutkan sebagai berikut.

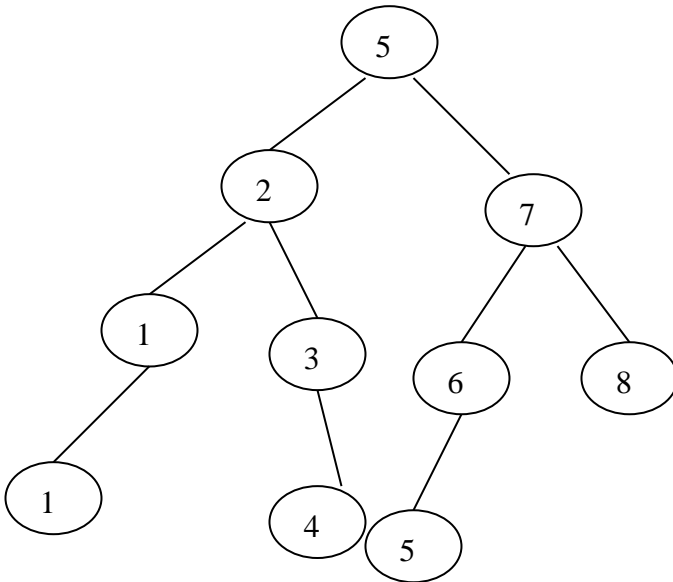
  p=root;
  Inorder(p," Root");
  Preorder(p," Root");
  Postorder(p," Root");
  cout<<endl<<"Cari apa? ";cin>>info.key;
  p=root;
  p=TreeSearch(p,info.key);
  if (p)
  {
    cout<<"Angka yang dicari "<<p->info.key<<endl;
    cout<<"Berada di posisi "<<p->info.posisi;
  }
  else cout<<"Tidak ada he..."<<endl;
  free(root);
  getch();
}
//-----

```

Jalankan dan amati hasilnya.

6.5. Soal Latihan

1. Terdapat pohon seperti dibawah, buatlah implementasi program untuk menciptakan pohon tersebut.



BAB VII

HASH TABLE

Capaian Pembelajaran

1. Mengetahui *hash table* dan *hashing*
2. Mengetahui kelebihan algoritma Hash
3. Memilih fungsi hash (*hash function*) dan metode - metode penanganan duplikasi kunci pada data atau tabrakan (*collision*) pada tabel hash
4. Kemampuan untuk membuat program yang mengimplementasikan konsep hash.

7.1. Teori Singkat Hash Table

Hash table atau tabel *hash* digunakan untuk mengindeks sekumpulan array untuk memudahkan proses pencarian. Sebagai contoh PIN perusahaan menggunakan 5 digit, rentang nilainya adalah 00000-99999. Saat menggunakan array, dapat menggunakan array yang berisi 100.000 elemen. Pada kenyataan perusahaan hanya memiliki 100 karyawan. Tentu saja, menggunakan 100.000 elemen akan menyebabkan banyak ruang yang tidak terpakai atau memori yang terbuang. Jadi berapa banyak array yang harus digunakan untuk mengisi data pencarian kunci dengan cepat? Tentu saja membutuhkan array kecil yang dapat menampung semua data. Jadi bagaimana memetakan antara lokasi PIN dan Array? Jawabannya adalah menggunakan fungsi hash (*hash function*).

Hash function adalah fungsi yang digunakan untuk mengubah nilai kunci menjadi nilai yang disebut alamat *hash*. Alamat *hash* ini mewakili indeks posisi dalam array. Teknologi yang dapat dengan mudah dan cepat memperoleh lokasi rekaman melalui fungsi hash disebut *hashing*, sehingga data karyawan dapat dicari tanpa membandingkan dengan

kunci lainnya. Array yang digunakan untuk menyimpan data melalui hashing nama tabel *hash*. Tabel *hash* biasanya disebut sebagai struktur data, yang digunakan untuk menyimpan data berdasarkan fungsi *hash*.

7.2. Memilih Fungsi Hash

Kriteria pemilihan fungsi hash pertama-tama harus mempertimbangkan bahwa perhitungannya harus mudah dan cepat, dan kedua, harus menghasilkan nilai yang didistribusikan dalam rentang indeks array. Metode yang digunakan untuk membentuk fungsi hash adalah sisa pembagian, pemotongan dan pelipatan (*folding*).

1. Sisa Pembagian

Konsep sisa pembagian adalah membagi nilai kunci, misalnya PIN pada data karyawan dibagi dengan nilai, dan sisa pembagian digunakan sebagai alamat hash. Secara matematis, fungsi hash ditulis sebagai:

$$H(k) = k \bmod m, m > n$$

dengan:

- k adalah kunci
- m adalah suatu bilangan pembagi
- n adalah jumlah data

Dengan asumsi bahwa mode $k \bmod m$ menghasilkan angka antara 0 dan $m-1$, maka jika lokasi memori (indeks array) dimulai dengan 1, maka hasil pembagian harus berjumlah 1.

$$H(k) = (k \bmod m) + 1$$

2. Pemotongan

Metode pemotongan dilakukan dengan mengabaikan beberapa bagian kunci dan menggunakan sisanya sebagai indeks untuk mengakses data dalam tabel *hash*.

3. Pelipatan (*Folding*)

Dalam metode *folding*, kunci dibagi menjadi beberapa bagian, seperti dua digit, dan kemudian ditambahkan. Hasilnya dipotong untuk memasukkan rentang indeks di tabel *hash*.

7.3. Menangani Tabrakan (*Collision*) Dalam Tabel Hash

Apapun metode yang digunakan bisa saja menimbulkan 2 buah kunci atau lebih diterjemahkan oleh fungsi hash ke dalam nilai yang sama. Situasi yang membuat beberapa kunci memiliki alamat hash yang sama atau disebut dengan tabrakan hash (*hash collision*). Untuk mengatasinya terdapat 3 teknik yaitu: Pengalamatan Terbuka (*Open Addressing*), Pembentukan rantai (*Chaining*) dan Pengalamatan Bucket (*bucket addressing*).

Apa pun metode yang digunakan, dapat mengonversi dua kunci atau lebih fungsi *hash* ke nilai yang sama. Pada kasus ini, beberapa kunci memiliki alamat hash yang sama disebut tabrakan *hash* (*hash collision*). Untuk mengatasi masalah ini, ada 3 teknologi, yaitu: pembentukan rantai (*chaining*) dan pengalamatan bucket (*bucket addressing*).

1. Pengalamatan Terbuka (*Open Addressing*)

Saat pengalamatan dibuka, semua elemen disimpan dalam tabel *hash* itu sendiri. Beberapa pemeriksaan yang menyertakan pengalamatan terbuka adalah sebagai berikut:

- **Pemeriksaan linear (*linear probing*)**

Tabrakan *hash* ditangani dengan menemukan lokasi kosong terdekat, yang disebut verifikasi *linier*. Mengenai fungsi *hash* untuk verifikasi *linier* sebagai berikut;

$$h(k,i) = (h'(k)+i) \bmod m, m=0..n-1$$

- **Pemeriksaan Kuadratik** yang dilakukan dengan urutan sbb:

$$h(k,i) = (h'(k)+i^2) \bmod m, i=0..n-1$$

Contoh urutan pencarian sbb:

$$h, h+1, h+4, h+9, \dots, h+i^2$$

- **Double hashing**, pemeriksaan dilakukan dengan urutan sbb:

$$h(k,i) = (h_1(k) + ih_2(k)) \bmod m$$

Dengan h_1 dan h_2 adalah fungsi hash contoh pemeriksaan dengan *double hashing*:

$$h_1, h_1+h_2, h_1+2h_2, h_1+3h_2, \dots, h_1+i \times h_2$$

Dalam pemeriksaan kuadratik maupun *double hashing*, h mewakili alamat *hash* yang diperoleh oleh fungsi *hash*, dan i dimulai dari 0.

2. Teknik Pembentukan rantai (*Chaining*)

Dalam teknik ini, data dalam tabel *hash* dibentuk secara dinamis menggunakan pembentukan rantai (*chaining*).

3. Teknik Pengalamatan Buket

Teknik ini mirip dengan pembentukan rantai (*chaining*), tetapi tabrakan tidak ditangani dengan teknik menggunakan *array*. Pengalamatan *buket* itu sendiri didefinisikan sebagai ruang memori sehingga cukup untuk menampung banyak data dengan alamat *hash* yang sama.

7.4. Praktik Hast Table

Buatlah sebuah project yang berisi File **HashLin.h**, **HashLin.cpp** dan **Main.cpp**

1. HashLin.h

```
#ifndef HASHLINEAR_H
#define HASHLINEAR_H
#include <string>
#include <vector>
using namespace std;
class DataHash{
public:
    int nip;
    string nama;

    // Konstruktor
    DataHash(int nip, string nama);
};
```

```

class HashLinear{
private:
    vector<DataHash*> data; // Vektor untuk tabel hash
    int ukuranTabel;      // Ukuran data dalam vektor
    DataHash* ptrTerhapus; // Menunjuk ke data yang dihapus
public:
    HashLinear(int ukuran);
    ~HashLinear();
    int hashFunction(int nip);
    void insert(int nip, string nama);
    string find(int nip);
    bool remove(int nip);
    void display();
};
#endif // HASHLINEAR_H

```

2. Program HashLin.cpp

```

#include <iostream>
#include <string>
#include "HashLin.h"
using namespace std;

DataHash::DataHash(int nip, string nama){
    DataHash::nip = nip;
    DataHash::nama = nama;
}

HashLinear::HashLinear(int ukuran){
    ukuranTabel = ukuran;
    data.resize(ukuran);
}

```

```

// Kosongkan tabel hash
for (int i = 0; i < ukuran; i++) {
    HashLinear::data[i] = NULL;
}
// Tandai NIP dengan -1 untuk menyatakan terhapus
ptrTerhapus = new DataHash(-1, "");
}

HashLinear::~~HashLinear(){
    for (int i = 0; i < ukuranTabel; i++) {
        if (data[i] != NULL)
            if (data[i]->nip != -1)
                delete data[i];
    }

    delete ptrTerhapus;
}
// hasFunction (publik) Menghasilkan alamat hash berdasarkan nip
int HashLinear::hashFunction(int nip)
{
    return nip % ukuranTabel;
}
// insert (publik) Untuk menyisipkan data ke tabel hash
void HashLinear::insert(int nip, string nama){
    int alamatHash = hashFunction(nip);
    int i = 0;
    while ((data[alamatHash] != NULL) &&
        (data[alamatHash]->nip != -1)) {
        if (i > ukuranTabel)
            break; // Keluar kalau sudah penuh
    }
}

```

```

// Peroleh alamat berikutnya
alamatHash = (alamatHash + 1) % ukuranTabel;
i++; // Untuk menghitung sampai penuh
}

if (i < ukuranTabel)
    data[alamatHash] = new DataHash(nip, nama);
}
// find() (publik). Mencari data. Nilai balik berupa data nama
string HashLinear::find(int nip){
    int alamatHash = hashFunction(nip);
    int i = 1;
    while (data[alamatHash] != NULL) {
        if (i > ukuranTabel)
            break; // Keluar kalau semua sudah dicek

        // Cek nilai field Nip sama dengan Nip
        if (data[alamatHash]->nip == nip)
            break; // Keluar while

        // Peroleh alamat berikutnya
        alamatHash = (alamatHash + 1) % ukuranTabel;
        i++; // Untuk menghitung sampai semua
    }
    if ((i <= ukuranTabel) &&
        (data[alamatHash] != NULL))
        return data[alamatHash]->nama;
    else
        return "<Tidak ditemukan>"; // Kalau tidak ditemukan
}
// remove (publik) false jika gagal menghapus Nip sebaliknya True

```

```
bool HashLinear::remove(int nip){
    int alamatHash = hashFunction(nip);
    int i = 1;
    while (data[alamatHash] != NULL) {
        if (i > ukuranTabel)
            break; // Keluar kalau semua sudah dicek

        // Cek nilai field Nip sama dengan Nip
        if (data[alamatHash]->nip == nip) {
            // Bebaskan memori yang digunakan untuk menyimpan data
            delete data[alamatHash];

            // Alihkan ke pointer yang menyatakan data terhapus
            data[alamatHash] = ptrTerhapus;
            break; // Keluar while
        }
        // Peroleh alamat berikutnya
        alamatHash = (alamatHash + 1) % ukuranTabel;
        i++; // Untuk menghitung sampai semua
    }

    if ((i < ukuranTabel) &&
        (data[alamatHash] != NULL))
        return true;
    else
        return false; // gagal menghapus
}

// display (publik) Menampilkan isi tabel hash
void HashLinear::display(){
    for (int i=0; i < ukuranTabel; i++) {
        if (data[i] == NULL)
```

```
        cout << i << ": (Kosong) " << endl;
    else
        if (data[i]->nip == -1)
            cout << i << ": (Terhapus) " << endl;
        else
            cout << i << ": "
                << data[i]->nip << "-"
                << data[i]->nama << endl;
    }
}
```

3. Main.cpp

```
#include <iostream>
#include "HashLin.h"

using namespace std;

int main()
{
    // Buat tabel hash untuk 11 data
    HashLinear tabel(11);

    // Memasukkan data ke dalam tabel hash
    tabel.insert(12, "Andika");
    tabel.insert(13, "Ratna");
    tabel.insert(16, "Dewi");
    tabel.insert(17, "Ari Wardi");
    tabel.insert(20, "Karmila");

    cout << "Keadaan awal: " << endl;
    tabel.display();
}
```

```
// Tes tabrakan hash
tabel.insert(23, "Diana Riana");
tabel.insert(45, "Shanti");
tabel.insert(108, "Ahmad Dani");

cout << "Setelah ditambah: " << endl;
tabel.display();

// Cari Data
cout << "Nip 17: "
    << tabel.find(17) << endl;

cout << "Nip 48: "
    << tabel.find(48) << endl;

// Hapus data
tabel.remove(17);

cout << "Keadaan setelah NIP 17 dihapus: " << endl;
tabel.display();

return 0;
```

7.5. Soal Latihan

1. Berikat contoh penggunaan *Hash table*
2. Metode apa yang digunakan untuk membentuk fungsi hash

DAFTAR PUSTAKA

- Kristanto, A., 2003, *Struktur Data dengan C++*, Graha Ilmu, Yogyakarta.
- Lafare, R., 1999, *Sams Teach Yourself Data Structures and Algorithms In 24 Hours*, Sams Publish



Muhammad Bahit, seorang pendidik berdedikasi di bidang Ilmu Komputer, telah menjadi dosen dalam program D3 Komputerisasi Akuntansi di Politeknik Negeri Banjarmasin sejak Mei 2018 hingga kini. Dengan latar belakang pendidikan yang kuat, beliau meraih gelar Magister Teknologi Informasi dari Universitas Gadjah Mada pada tahun 2013, menambah dimensi mendalam pada wawasan dan pengetahuannya.

Pendidikan awalnya melibatkan studi Manajemen Informatika di Politeknik Negeri Banjarmasin pada tahun ajaran 2008, memberikan dasar yang kokoh untuk pengembangan karirnya di dunia teknologi informasi. Sebagai dosen, Muhammad Bahit telah membimbing mahasiswa dengan penuh semangat, memberikan wawasan yang luas dan pemahaman yang mendalam di bidangnya.

Muhammad Bahit memiliki keahlian dalam mengajar dan membagikan pengetahuan terkini mengenai komputerisasi akuntansi, menciptakan lingkungan belajar yang mendukung perkembangan kritis dan kreatif para mahasiswa. Di samping itu, ketertarikannya dalam Magister Teknologi Informasi mencerminkan dedikasinya terhadap pengembangan diri dan pemahaman mendalam terhadap perkembangan terkini di dunia teknologi.

Sebagai seorang akademisi yang berpengalaman, Muhammad Bahit terus berusaha untuk menggabungkan teori dengan praktik dalam pendekatannya mengajar. Dengan memadukan keahlian akademis dan wawasan praktisnya, beliau bertujuan untuk mengilhami para mahasiswa dan membantu mereka mempersiapkan diri untuk kesuksesan di dunia teknologi informasi yang terus berkembang.

Algoritma Pemrograman Terstruktur



Mudah-mudahan buku ajar ini bisa membantu mahasiswa untuk memahami secara kontekstual. Penulis yakin bahwa materi dalam bahan kuliah ini masih jauh dari kesempurnaan, sehingga terbuka untuk mendapatkan kritik dan saran untuk perbaikan pada semua sisi penulisannya.

Capaian Pembelajaran

1. Memahami tentang penggunaan tipe data larik dan pointer.
2. Menggunakan tipe data untuk menyelesaikan soal.
3. Memahami tentang sorting dengan metode insertion sort dan selection sort.

Muhammad Bahit



Penerbit Poliban Press

Redaksi :

Politeknik Negeri Banjarmasin, Jl. Brigjen H. Hasan Basry,
Pangeran, Komp. Kampus ULM, Banjarmasin Utara

Telp. : (0511)3305052

Email : press@poliban.ac.id

ISBN 978-623-5259-10-9 (PDF)

