

# **MODUL PRAKTIKUM “STRUKTUR DATA”**



**Bahasa Pemrograman : C++**

**Software : Turbo C++ 4.5**

**Laboran : M. Fachrurrozi**

**Novi Yusliani**

**LABORATORIUM DASAR KOMPUTER  
PROGRAM ILMU KOMPUTER  
UNIVERSITAS SRIWIJAYA  
2006**

# DAFTAR ISI

DAFTAR ISI .....	1
Bab 1. POINTER .....	2
Bab 2. ARRAY.....	5
2.1. Array Satu Dimensi .....	5
2.2. Array Dua Dimensi.....	7
Bab 3. STRUCTURE.....	16
Bab 4. LINKED LIST.....	19
4.3. Double Linked List .....	22
Bab 5. STACK.....	30
5.1. Definisi Stack.....	30
5.2. Stack dengan Array .....	31
5.3. Double Stack dengan Array .....	34
5.4. Stack dengan Single Linked List.....	35
Bab 6. QUEUE.....	40
6.1. Definisi Queue.....	40
6.2. Implementasi Queue dengan Linear Array .....	40
6.3. Implementasi Queue dengan Circular Array .....	42
6.4. Implementasi Queue dengan Double Linked List .....	43
Bab 7. TREE.....	50
7.1. Definisi Tree .....	50
7.2. Jenis-Jenis Tree .....	51
7.3. Operasi-Operasi pada Binary Tree .....	52
7.4. Binary Search Tree .....	53
REFERENSI.....	62

## Bab 1. POINTER

Pointer merupakan tipe data berukuran 32 bit yang berisi satu nilai yang berpadanan dengan alamat memori tertentu. Sebagai contoh, sebuah variabel P bertipe pointer bernilai 0x0041FF2A, berarti P menunjuk pada alamat memori 0041FF2A. Pointer dideklarasikan seperti variabel biasa dengan menambahkan tanda \* (asterik) yang mengawali nama variabel.

Bentuk Umum:

<tipe data> namaVariabel;

Contoh:

float \* px;

Statement di atas mendeklarasikan variabel px yang merupakan pointer. Penyebutan tipe data float berarti bahwa alamat memori yang ditunjuk oleh px dimaksudkan untuk berisi data bertipe float.

Contoh Program:

1:

```

#include <iostream.h>

void main()
{
    int x;
    int *px;

    x = 2;
    px = &x;    //membaca alamat dari x

    cout<<"Nilai x          : "<<x<<endl;
    cout<<"Nilai *px        : "<<x<<endl;
    cout<<"Nilai px (alamat x) : "<<px<<endl;
}

```

Output:

```

Nilai x          : 2
Nilai *px        : 2
Nilai px (alamat x) : 0x2467242e

```

2:

```

#include <iostream.h>

void main()
{
    int x[10]={0,1,2,3,4,5,6,7,8,9};
    int *px;
    int i;

    for (i=0;i<10;i++)
    {
        px = &x[i];    //membaca alamat dari x

        cout<<x[i]<<" "<<*px<<" "<<px<<endl;
    }
}

```

Output:

```
0 0 0x250f23da
1 1 0x250f23dc
2 2 0x250f23de
3 3 0x250f23e0
4 4 0x250f23e2
5 5 0x250f23e4
6 6 0x250f23e6
7 7 0x250f23e8
8 8 0x250f23ea
9 9 0x250f23ec
```

3:

```
#include <iostream.h>

void main()
{
    char *nama;

    nama = "Muhammad Fachrurrozi";
    cout<<"Selamat datang "<<nama<<endl;
}
```

Output:

```
Selamat datang Muhammad Fachrurrozi
```

## Bab 2. ARRAY

Array adalah suatu struktur yang terdiri dari sejumlah elemen yang memiliki tipe data yang sama. Elemen-elemen array tersusun secara sekuensial dalam memori komputer. Array dapat berupa satu dimensi, dua dimensi, tiga dimensi ataupun banyak dimensi (multi dimensi).

### 2.1. Array Satu Dimensi

Array Satu dimensi tidak lain adalah kumpulan elemen-elemen identik yang tersusun dalam satu baris. Elemen-elemen tersebut memiliki tipe data yang sama, tetapi isi dari elemen tersebut boleh berbeda.

Elemen ke-	0	1	2	3	4	5	6	7	8	9
Nilai	23	34	32	12	25	14	23	12	11	10

Bentuk umum:

<tipe data> NamaArray[n] = {elemen0, elemen1, elemen2,.....,n};

n = jumlah elemen

Contoh Program:

1.

```
/*Program Mencari bilangan terkecil
dan terbesar di dalam array */

#include <iostream.h>

void main()
{
    int x[10]={45,34,23,34,32,12,65,76,34,23};
    int i;
    int maks = -1000; //asumsi paling minimum
    int min = 1000;  //asumsi paling maksimum
```

```

    for (i=0; i<10; i++)
    {
        if (x[i]>maks)
        {
            maks = x[i];
        }

        if (x[i]<min)
        {
            min = x[i];
        }
    }

    cout<<"Nilai maksimum : "<<maks<<endl;
    cout<<"Nilai minimum : "<<min<<endl;
}

```

Output:

```

Nilai maksimum : 76
Nilai minimum : 12

```

2.

```

/*Program Mencari bilangan tertentu di dalam array
mencari jumlah data yang ditemukan dan
di elemen mana saja bilangan itu ditemukan */

#include <iostream.h>
typedef enum {false=0, true=1} bool;

void main()
{
    int x[10]={45,34,23,34,32,12,65,76,34,23};
    int i,bil,jumlah;
    bool ketemu=false;

    jumlah = 0;

    cout<<"Bilangan yang akan dicari : ";
    cin>>bil;

```

```

for (i=0;i<10;i++)
{
    if (x[i]==bil)
    {
        ketemu = true;
        cout<<"Bilangan ditemukan di elemen : "<<i<<endl;
        jumlah++;
    }
}

if (ketemu)
{
    cout<<"Jumlah data : "<<jumlah;
}
else
{
    cout<<"Bilangan tersebut tidak ditemukan";
}
}

```

Output:

```

Bilangan yang akan dicari : 23
Bilangan ditemukan di elemen : 2
Bilangan ditemukan di elemen : 9
Jumlah data : 2

```

Atau

```

Bilangan yang akan dicari : 30
Bilangan tersebut tidak ditemukan

```

## 2.2. Array Dua Dimensi

Array dua dimensi sering digambarkan sebagai sebuah matriks, merupakan perluasan dari array satu dimensi. Jika *array satu dimensi hanya terdiri dari sebuah baris* dan beberapa kolom elemen, maka *array dua dimensi terdiri dari beberapa baris* dan beberapa kolom elemen yang bertipe sama sehingga dapat digambarkan sebagai berikut:



	0	1	2	3	4	5	6
0	10	21	23	43	45	78	65
1	45	43	65	12	21	12	21
2	32	34	23	56	54	34	45
3	11	12	32	23	56	76	45

Bentuk umum:

`<type data> NamaArray [m][n];`

Atau

`<type data> NamaArray [m][n] = { {a,b,...z},{1,2,...,n-1} };`

Contoh:

`double matrix[4][4];`

`bool papan[2][2] = { {true,false},{true,false} };`

Pendeklarasian array dua dimensi hampir sama dengan pendeklarasian array satu dimensi, kecuali bahwa array dua dimensi terdapat dua jumlah elemen yang terdapat di dalam kurung siku dan keduanya boleh tidak sama.

Elemen array dua dimensi diakses dengan menuliskan kedua indeks elemennya dalam kurung siku seperti pada contoh berikut:

`//papan nama memiliki 2 baris dan 5 kolom`

`bool papan[2][5];`

`papan[0][0] = true;`

`papan[0][4] = false;`

`papan[1][2] = true;`

`papan[1][4] = false;`

## Contoh program:

1.

```
/*Penjumlahan 2 Buah Matriks 2x2*/

#include <iostream.h>
#include <conio.h> //untuk mengaktifkan perintah gotoxy(x,y) dan clrscr()

#define Nmaks 10

typedef int matrik[Nmaks][Nmaks];

void main()
{
    int n,i,j;
    matrik A,B,C;

    cout<<"Program Penjumlahan Matrik A 2x2 dan B 2x2"<<endl;
    cout<<endl;

    n=2;
    cout<<"Masukkan Nilai-Nilai Matrik A"<<endl;
    for (i=1; i<=n; i++)
    {
        for (j=1; j<=n; j++)
        {
            cout<<"A["<<i<<","<<j<<"] = ";
            cin>>A[i][j];
        }
    }

    clrscr();
    cout<<"Masukkan Nilai-Nilai Matrik B"<<endl;
    for (i=1; i<=n; i++)
    {
        for (j=1; j<=n; j++)
        {
            cout<<"B["<<i<<","<<j<<"] = ";
            cin>>B[i][j];
        }
    }
}
```

```

clrscr();
cout<<endl;
//Proses Penjumlahan Matrik C = A + B
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        C[i][j] = A[i][j] + B[i][j];
    }
}

clrscr();
cout<<"Nilai-Nilai Matriks A, B, dan C"<<endl;
cout<<endl;

//Proses Output Matrik A
gotoxy(1,5);
cout<<"A = ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(2+4*j,2+2*i);
        cout<<A[i][j];
    }
}

//Proses Output Matrik B
gotoxy(1,10);
cout<<"B = ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(2+4*j,7+2*i);
        cout<<B[i][j];
    }
}

```

```

//Proses Output Matrik C
gotoxy(1,15);
cout<<"C = ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(2+4*j, 12+2*i);
        cout<<A[i][j];
    }
}

gotoxy(12,15);
cout<<" + ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(13+4*j, 12+2*i);
        cout<<B[i][j];
    }
}

gotoxy(23,15);
cout<<" = ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(24+4*j, 12+2*i);
        cout<<C[i][j];
    }
}
}

```

Output:

Program Penjumlahan Matrik A 2x2 dan B 2x2

Masukkan Nilai-Nilai Matrik A

A[1,1] = 1  
A[1,2] = 2  
A[2,1] = 5  
A[2,2] = 8\_

Masukkan Nilai-Nilai Matrik B

B[1,1] = 5  
B[1,2] = 1  
B[2,1] = 6  
B[2,2] = 4

Nilai-Nilai Matriks A, B, dan C

A =  
1 2  
5 8

B =  
5 1  
6 4

C =  
1 2 + 5 1 = 6 3  
5 8 6 4 11 12

2.

```
/*Perkalian 2 Buah Matriks 2x2*/  
  
#include <iostream.h>  
#include <conio.h> //untuk mengaktifkan perintah gotoxy(x,y) dan clrscr()  
  
#define Nmaks 10  
  
typedef int matrik[Nmaks][Nmaks];  
  
void main()  
{  
    int n,i,j;  
    matrik A,B,C;  
  
    cout<<"Program Perkalian Matrik A 2x2 dan B 2x2"<<endl;  
    cout<<endl;  
  
    n=2;  
    cout<<"Masukkan Nilai-Nilai Matrik A"<<endl;  
    for (i=1; i<=n; i++)  
    {  
        for (j=1; j<=n; j++)  
        {  
            cout<<"A["<<i<<","<<j<<"] = ";  
            cin>>A[i][j];  
        }  
    }  
}
```

```

clrscr();
cout<<"Masukkan Nilai-Nilai Matrik B"<<endl;
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        cout<<"B["<<i<<","<<j<<"] = ";
        cin>>B[i][j];
    }
}

clrscr();
cout<<endl;
//Proses Penjumlahan Matrik C = A + B
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        C[i][j] = (A[i][1]*B[1][j]) + (A[i][2]*B[2][j]);
    }
}

clrscr();
cout<<"Nilai-Nilai Matriks A, B, dan C"<<endl;
cout<<endl;

//Proses Output Matrik A
gotoxy(1,5);
cout<<"A = ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(2+4*j,2+2*i);
        cout<<A[i][j];
    }
}

//Proses Output Matrik B
gotoxy(1,10);
cout<<"B = ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(2+4*j,7+2*i);
        cout<<B[i][j];
    }
}

```

```

//Proses Output Matrik C
gotoxy(1,15);
cout<<"C = ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(2+4*j,12+2*i);
        cout<<A[i][j];
    }
}

gotoxy(12,15);
cout<<" x ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(13+4*j,12+2*i);
        cout<<B[i][j];
    }
}

gotoxy(23,15);
cout<<" = ";
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        gotoxy(24+4*j,12+2*i);
        cout<<C[i][j];
    }
}
}

```

Output:

Program Perkalian Matrik A 2x2 dan B 2x2

Masukkan Nilai-Nilai Matrik A

A[1,1] = 1  
A[1,2] = 5  
A[2,1] = 4  
A[2,2] = 9\_

Masukkan Nilai-Nilai Matrik B

B[1,1] = 2  
B[1,2] = 6  
B[2,1] = 4  
B[2,2] = 1

---

Nilai-Nilai Matriks A, B, dan C

$$A = \begin{pmatrix} 1 & 5 \\ 4 & 9 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 6 \\ 4 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 5 \\ 4 & 9 \end{pmatrix} \times \begin{pmatrix} 2 & 6 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 22 & 11 \\ 44 & 33 \end{pmatrix}$$

Latihan:

1. Buat program menghitung penjumlahan matrik 3x3.
2. Buat program menghitung perkalian matrik 3x3.



## Bab 3. STRUCTURE

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

Field-field tersebut digabungkan menjadi satu dengan tujuan untuk kemudahan dalam operasinya. Misalnya Anda ingin mencatat data-data mahasiswa dan pelajar dalam sebuah program, Untuk membedakannya Anda dapat membuat sebuah record mahasiswa yang terdiri dari field nim, nama, alamat dan ipk serta sebuah record pelajar yang terdiri dari field-field nama, nonurut, alamat dan jumnilai. Dengan demikian akan lebih mudah untuk membedakan keduanya.

Bentuk umum:

```
struct namastruct
{
    <tipe data> field1;
    <tipe data> field2;
    <tipe data> field3;
};
```

Contoh:

```
struct mahasiswa
{
    char nim[11];
    char nama[30];
    char alamat[50];
    float ipk;
};
```

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

```
cout<<mahasiswa.nim;
```

Jika Pmhs adalah pointer bertipe mahasiswa\* maka field dari Pmhs dapat diakses dengan mengganti tanda titik dengan tanda panah (" à ").

```
cout<<mahasiswa->nim;
```

Contoh program:

1.

```
/* Mengisi Biodata dan Nilai IPK mahasiswa */
#include <iostream.h>

struct mahasiswa
{
    char nim[15];
    char nama[30];
    char alamat[50];
    float ipk;
};

void main()
{
    mahasiswa mhs;

    cout<<"NIM      : "; cin.getline(mhs.nim,15);
    cout<<"Nama      : "; cin.getline(mhs.nama,30);
    cout<<"Alamat    : "; cin.getline(mhs.alamat,50);
    cout<<"Nilai IPK  : "; cin>>mhs.ipk;

    cout<<endl;
```

```

        cout<<endl;

        cout<<"NIM Anda          : "<<mhs.nim<<endl;
        cout<<"Nama Anda         : "<<mhs.nama<<endl;
        cout<<"Alamat Anda        : "<<mhs.alamat<<endl;
        cout<<"Nilai IPK Anda     : "<<mhs.ipk<<endl;
    }

```

#### Output:

```

NIM          : 09983110077
Nama         : M. Fachrurrozi
Alamat       : Jl. Joko Atas No 23 Palembang
Nilai IPK    : 3.00

NIM Anda     : 09983110077
Nama Anda    : M. Fachrurrozi
Alamat Anda  : Jl. Joko Atas No 23 Palembang
Nilai IPK Anda : 3

```

#### Latihan:

1. Buat program menghitung durasi rental warnet, dengan ketentuan perhitungannya:

30 detik = Rp. 130,-

Satuan waktu : jam : menit : detik

2. Buat program menghitung jumlah nilai akhir mahasiswa dengan ketentuan:

Nilai akhir = 10%\*tugas + 20%\*kuis + 30%\*mid + 40%\*uas

Nilai Huruf:

Nilai akhir >85	: A
85 >= nilai akhir > 70	: B
70 >= nilai akhir > 55	: C
55 >= nilai akhir > 40	: D
Nilai akhir <=40	: E

## Bab 4. LINKED LIST

Pada bab sebelumnya telah dijelaskan mengenai variabel array yang bersifat statis (ukuran dan urutannya sudah pasti). Selain itu, ruang memori yang dipakai olehnya tidak dapat dihapus bila array tersebut sudah tidak digunakan lagi pada saat program dijalankan. Untuk memecahkan masalah di atas, kita dapat menggunakan variabel pointer. Tipe data pointer bersifat dinamis, variabel akan dialokasikan hanya pada saat dibutuhkan dan sesudah tidak dibutuhkan dapat direlokasikan kembali.

Setiap ingin menambahkan data, Anda selalu menggunakan variabel pointer yang baru, akibatnya Anda akan membutuhkan banyak sekali pointer. Oleh karena itu, ada baiknya jika Anda hanya menggunakan satu variabel pointer saja untuk menyimpan banyak data dengan metode yang kita sebut Linked List. Linked list adalah sekumpulan elemen bertipe sama, yang mempunyai keterurutan tertentu, yang setiap elemennya terdiri dari dua bagian.

Bentuk Umum :

```
typedef struct telmtlist
{
    infotype info;
    address next;
}elmtlist;
```

infotype   8   sebuah tipe terdefinisi yang menyimpan informasi sebuah elemen list

next       8   address dari elemen berikutnya (suksesor)

Jika L adalah list, dan P adalah address, maka alamat elemen pertama list L dapat diacu dengan notasi :

`first(L)`

Sebelum digunakan harus dideklarasikan terlebih dahulu :

```
#define first(L) (L)
```

Elemen yang diacu oleh P dapat dikonsultasi informasinya dengan notasi :

```
info(P)    deklarasi  #define info(P) (P)->info
```

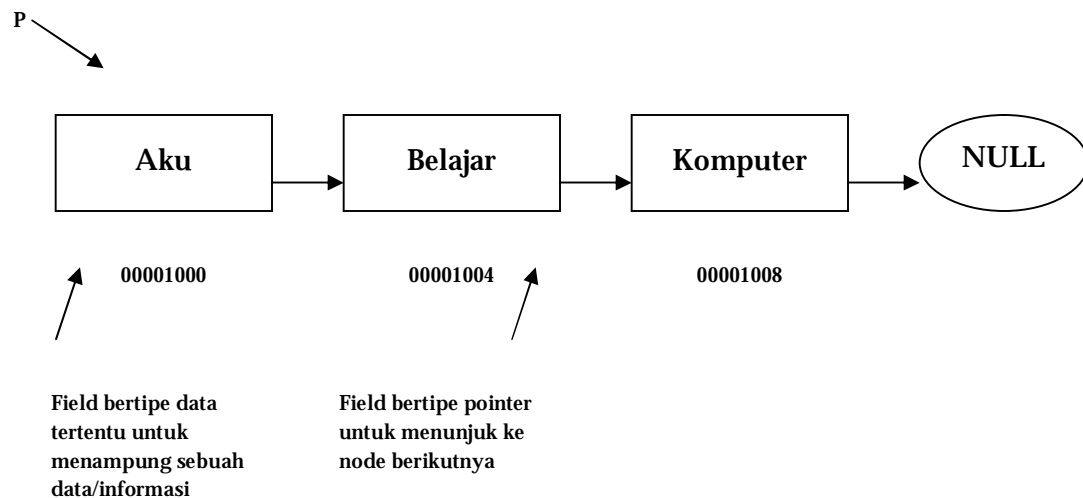
```
next(P)    deklarasi  #define next(P) (P)->next
```

Beberapa Definisi :

1. List l adalah list kosong, jika  $\text{First}(L) = \text{Nil}$
2. Elemen terakhir dikenali, dengan salah satu cara adalah karena  
 $\text{Next}(\text{Last}) = \text{Nil}$

Nil adalah pengganti Null, perubahan ini dituliskan dengan `#define Nil Null`

#### 4.1. Single Linked List



Pada gambar di atas tampak bahwa sebuah data terletak pada sebuah lokasi memori area. Tempat yang disediakan pada satu area memori tertentu untuk menyimpan data dikenal dengan sebutan node/simpul. Setiap node memiliki pointer yang menunjuk ke simpul berikutnya sehingga terbentuk satu untaian, dengan demikian hanya diperlukan sebuah variabel pointer. Susunan berupa untaian semacam ini disebut Single Linked List (NULL memiliki nilai khusus yang artinya tidak menunjuk ke mana-mana. Biasanya Linked List pada titik akhirnya akan menunjuk ke NULL).

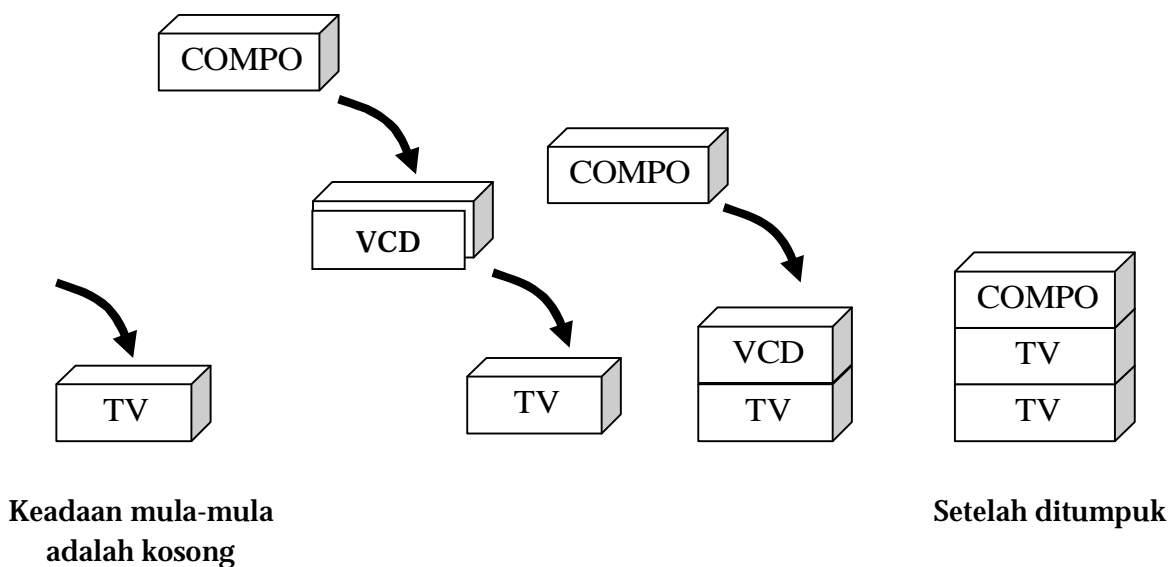
Pembuatan Single Linked List dapat menggunakan 2 metode:

- LIFO (Last In First Out), aplikasinya : Stack (Tumpukan)
- FIFO (First In First Out), aplikasinya : Queue (Antrean)

### LIFO ( Last In First Out)

Lifo adalah suatu metode pembuatan Linked List di mana data yang masuk paling akhir adalah data yang keluar paling awal. Hal ini dapat dianalogikan (dalam kehidupan sehari-hari) dengan saat Anda menumpuk barang seperti digambarkan dibawah ini.

Pembuatan sebuah simpul dalam suatu linked list seperti digambarkan dibawah ini. Jika linked list dibuat dengan metode LIFO, terjadi penambahan / Insert simpul di belakang, dikenal dengan istilah INSERT.



**Gambar. Ilustrasi Single Linked List dengan metode LIFO**

### FIFO (Fisrt In Fisrt Out)

FIFO adalah suatu metode pembuatan Linked List di mana data yang masuk paling awal adalah data yang keluar paling awal juga. Hal ini dapat di analogikan (dalam kehidupan sehari-hari), misalnya saat sekelompok orang yang datang (ENQUEUE) mengantri hendak membeli tiket di loket.

Jika linked list dibuat dengan metode FIFO, terjadi penambahan / Insert simpul didepan.

### ***4.3. Double Linked List***

Salah satu kelemahan single linked list adalah pointer (penunjuk) hanya dapat bergerak satu arah saja, maju/ mundur, atau kanan/kiri sehingga pencarian data pada single linked list hanya dapat bergerak dalam satu arah saja. Untuk mengatasi kelemahan tersebut, anda dapat menggunakan metode double linked list. Linked list ini dikenal dengan nama Linked list berpointer Ganda atau Double Linked List.

### ***4.4. Circular Double Linked List***

Ini adalah double linked list yang simpul terakhirnya menunjuk ke simpul terakhirnya menunjuk ke simpul awalnya menunjuk ke simpul akhir sehingga membentuk suatu lingkaran.

Operasi-Operasi yang ada pada Linked List

#### ***Insert***

Istilah Insert berarti menambahkan sebuah simpul baru ke dalam suatu linked list.

#### ***IsEmpty***

Fungsi ini menentukan apakah linked list kosong atau tidak.

#### ***Find First***

Fungsi ini mencari elemen pertama dari linked list

#### ***Find Next***

Fungsi ini mencari elemen sesudah elemen yang ditunjuk now.

### ***Retrieve***

Fungsi ini mengambil elemen yang ditunjuk oleh now. Elemen tersebut lalu dikembalikan oleh fungsi.

### ***Update***

Fungsi ini mengubah elemen yang ditunjuk oleh now dengan isi dari sesuatu.

### ***Delete Now***

Fungsi ini menghapus elemen yang ditunjuk oleh now. Jika yang dihapus adalah elemen pertama dari linked list (head), head akan berpindah ke elemen berikut.

### ***Delete Head***

Fungsi ini menghapus elemen yang ditunjuk head. Head berpindah ke elemen sesudahnya.

### ***Clear***

Fungsi ini menghapus linked list yang sudah ada. Fungsi ini wajib dilakukan bila anda ingin mengakhiri program yang menggunakan linked list. Jika anda melakukannya, data-data yang dialokasikan ke memori pada program sebelumnya akan tetap tertinggal di dalam memori.



## Contoh Program:

### 1. Membuat Single Linked List

---

```
#include <iostream.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>

#define Nil NULL
#define info(P) P->info
#define next(P) P->next
#define First(L) (L)

typedef int InfoType;
typedef struct telmtlist *address;
typedef struct telmtlist
{
    InfoType info;
    address next;
}elmtlist;

typedef address list;

void CiptaSenarai(list *L)
{
    First(*L) = Nil;
}

list NodBaru(int m)
{
    list n;
    n = (list) malloc(sizeof(elmtlist));
    if (n != NULL)
    {
        info(n) = m;
        next(n) = Nil;
    }

    return n;
}
```

```

void SisipSenarai (list *L, list t, list p)
{
    if (p == Nil)
    {
        t->next = *L;
        *L = t;
    }
    else
    {
        t->next = p->next;
        p->next = t;
    }
}

void CetakSenarai (list L)
{
    list ps;
    for (ps=L; ps!=Nil; ps=ps->next)
    {
        cout<<" "<<info(ps)<<" -->";
    }
    cout<<" NULL"<<endl;
}

int main()
{
    list pel;
    list n;
    int i,k,nilai;

    CiptaSenarai(&pel);
    cout<<"Masukkan Banyak Data = ";
    cin>>k;
    for (i=1; i<=k; i++)
    {
        cout<<"Masukkan Data Senarai ke-"<<i<<" = ";
        cin>>nilai;
        n = NodBaru(nilai);
        SisipSenarai (&pel, n, NULL);
    }

    CetakSenarai(pel);
    return 0;
}

```

Output:

```
Masukkan Banyak Data = 7
Masukkan Data Senarai ke-1 = 6
Masukkan Data Senarai ke-2 = 2
Masukkan Data Senarai ke-3 = 6
Masukkan Data Senarai ke-4 = 1
Masukkan Data Senarai ke-5 = 7
Masukkan Data Senarai ke-6 = 5
Masukkan Data Senarai ke-7 = 8
8 --> 5 --> 7 --> 1 --> 6 --> 2 --> 6 --> NULL
```

## 2. Pencarian Nilai Terkecil dan Nilai Terbesar dalam sebuah Single Linked List

```
#include <iostream.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>

#define Nil NULL
#define info(P) P->info
#define next(P) P->next
#define First(L) (L)

typedef int InfoType;
typedef struct telmtlist *address;
typedef struct telmtlist
{
    InfoType info;
    address next;
}telmtlist;

typedef address list;

void CiptaSenarai(list *L)
{
    First(*L) = Nil;
}
```

```

list NodBaru(int m)
{
    list n;
    n = (list) malloc(sizeof(elmtlist));
    if (n != NULL)
    {
        info(n) = m;
        next(n) = Nil;
    }

    return n;
}

void SisipSenarai (list *L, list t, list p)
{
    if (p == Nil)
    {
        t->next = *L;
        *L = t;
    }
    else
    {
        t->next = p->next;
        p->next = t;
    }
}

void CetakSenarai (list L)
{
    list ps;
    for (ps=L; ps!=Nil; ps=ps->next)
    {
        cout<<" "<<info(ps)<<" -->";
    }
    cout<<" NULL"<<endl;
}

```

```

InfoType Max(list L)
{
    address Pmax,Pt;

    Pmax = First(L);

    if (next(Pmax) == Nil)
    {
        return (info(Pmax));
    }
    else
    {
        Pt = next(Pmax);
        while (Pt != Nil)
        {
            if (info(Pmax) < info(Pt))
            {
                Pmax = Pt;
            }
            else
            {
                Pt = next(Pt);
            }
        }

        return (info(Pmax));
    }
}

InfoType Min(list L)
{
    address Pmin,Pt;

    Pmin = First(L);

    if (next(Pmin) == Nil)
    {
        return (info(Pmin));
    }
    else
    {
        Pt = next(Pmin);
        while (Pt != Nil)
        {
            if (info(Pmin) > info(Pt))
            {
                Pmin = Pt;
            }
        }
    }
}

```

```

        else
        {
            Pt = next(Pt);
        }
    }

    return (info(Pmin));
}

void main()
{
    list pel;
    list n;
    int i,k,nilai,maks,min;

    CiptaSenarai(&pel);
    cout<<"Masukkan Banyak Data = ";
    cin>>k;
    for (i=1; i<=k; i++)
    {
        cout<<"Masukkan Data Senarai ke-"<<i<<" = ";
        cin>>nilai;
        n = NodBaru(nilai);
        SisipSenarai (&pel, n, NULL);
    }

    cout<<endl;
    CetakSenarai(pel);
    maks = Max(pel);
    min = Min(pel);

    cout<<endl;
    cout<<"Nilai Terbesar : "<<maks;
    cout<<endl;
    cout<<"Nilai Terkecil : "<<min;
}

```

Output:

```

Masukkan Banyak Data = 5
Masukkan Data Senarai ke-1 = 3
Masukkan Data Senarai ke-2 = 11
Masukkan Data Senarai ke-3 = 54
Masukkan Data Senarai ke-4 = 1
Masukkan Data Senarai ke-5 = 26

26 --> 1 --> 54 --> 11 --> 3 --> NULL

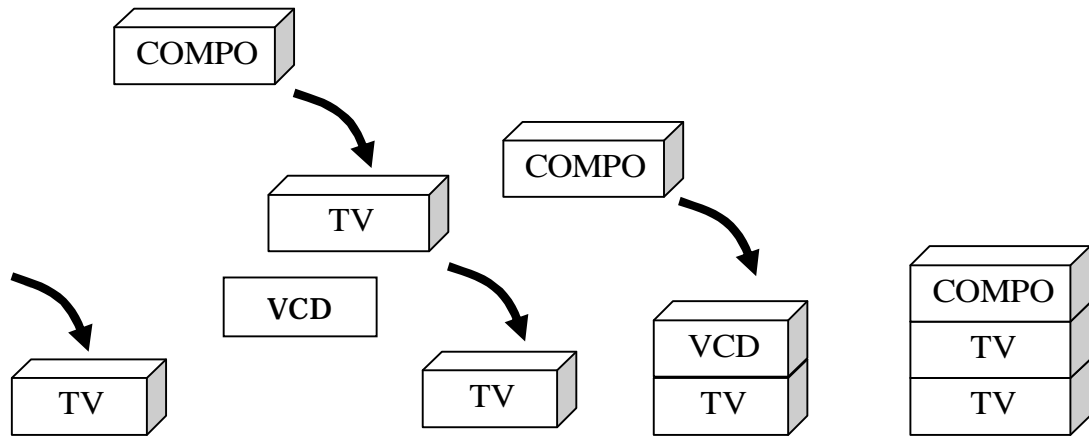
Nilai Terbesar : 54
Nilai Terkecil : 1

```

## Bab 5. STACK

### 5.1. Definisi Stack

Stack adalah suatu tumpukan dari benda. Konsep utamanya adalah LIFO (Last In First Out), benda yang terakhir masuk dalam stack akan menjadi benda pertama yang dikeluarkan dari stack.



Keadaan mula-mula  
adalah kosong

Setelah ditumpuk

Pada gambar di atas, jika kita ingin mengambil sesuatu dari tumpukan maka kita harus mengambil benda paling atas dahulu, yakni compo. Misalnya jika VCD langsung diambil, compo akan jatuh. Prinsip stack ini bisa diterapkan dalam pemrograman. Di C++, ada dua cara penerapan prinsip stack, yakni dengan array dan linked list. Setidaknya stack haruslah memiliki operasi-operasi sebagai berikut.

Push	Untuk menambahkan item pada tumpukan paling atas
Pop	Untuk mengambil item teratas
Clear	Untuk mengosongkan stack
IsEmpty	Untuk memeriksa apakah stack kosong
IsFull	Untuk memeriksa apakah stack sudah penuh
Retrieve	Untuk mendapatkan nilai dari item teratas

## ***5.2. Stack dengan Array***

Sesuai dengan sifat stack, pengambilan / penghapusan di elemen dalam stack harus dimulai dari elemen teratas.

Operasi-operasi pada Stack dengan Array

### ***IsFull***

Fungsi ini memeriksa apakah stack yang ada sudah penuh. Stack penuh jika puncak stack terdapat tepat di bawah jumlah maksimum yang dapat ditampung stack atau dengan kata lain  $Top = MAX\_STACK - 1$ .

### ***Push***

Fungsi ini menambahkan sebuah elemen ke dalam stack dan tidak bisa dilakukan lagi jika stack sudah penuh.

### ***IsEmpty***

Fungsi menentukan apakah stack kosong atau tidak. Tanda bahwa stack kosong adalah Top bernilai kurang dari nol.

### ***Pop***

Fungsi ini mengambil elemen teratas dari stack dengan syarat stack tidak boleh kosong.

### ***Clear***

Fungsi ini mengosongkan stack dengan cara mengeset Top dengan -1. Jika Top bernilai kurang dari nol maka stack dianggap kosong.

### ***Retreive***

Fungsi ini untuk melihat nilai yang berada pada posisi tumpukan teratas.



### Contoh Program :

#### Program untuk Insert (Push) Nilai dan Delete (Pop) Nilai dalam Stack

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX 10 //Ukuran Maksimum dari Stack

void push(int stack[], int *top, int value);
void pop(int stack[], int *top, int *value);

void main()
{
    int stack[MAX];
    int top = -1;
    int n, value;

    do
    {
        do
        {
            cout<<"Masukkan Nilai yang akan di Push : ";
            cin>>value;
            push(stack, &top, value);

            cout<<"Tekan 1 untuk Melanjutkan"<<endl;
            cin>>n;
        } while (n == 1);

        cout<<"Tekan 1 untuk Melakukan Pop"<<endl;
        cin>>n;

        while (n == 1)
        {
            pop(stack, &top, &value);
            cout<<"Nilai yang di Pop : "<<value;
            cout<<"Tekan 1 untuk Melakukan Pop sebuah Elemen"<<endl;
            cin>>n;
        }

        cout<<endl;
        cout<<"Tekan 1 untuk Melanjutkan"<<endl;
        cin>>n;
    } while (n == 1);
}
```

```

void push(int stack[], int *top, int value) //Fungsi untuk Insert Nilai
{
    if(*top < MAX)
    {
        *top = *top + 1;
        stack[*top] = value;
    }
    else
    {
        cout<<"Stack Penuh, Push Nilai Tidak Dapat Dilakukan"<<endl;
        exit(0);
    }
}

void pop(int stack[], int *top, int *value) //Fungsi untuk Delete Nilai
{
    if (*top >= 0)
    {
        *value = stack[*top];
        *top = *top - 1;
    }
    else
    {
        cout<<"Stack Kosong, Pop Tidak Dapat Dilakukan"<<endl;
        exit(0);
    }
}

```

Output :

```

Masukkan Nilai yang akan di Push : 12
Tekan 1 untuk Melanjutkan
1
Masukkan Nilai yang akan di Push : 23
Tekan 1 untuk Melanjutkan
1
Masukkan Nilai yang akan di Push : 14
Tekan 1 untuk Melanjutkan
1
Masukkan Nilai yang akan di Push : 25
Tekan 1 untuk Melanjutkan
2
Tekan 1 untuk Melakukan Pop
1
Nilai yang di Pop : 25
Tekan 1 untuk Melakukan Pop sebuah Elemen
1
Nilai yang di Pop : 14
Tekan 1 untuk Melakukan Pop sebuah Elemen
2

Tekan 1 untuk Melanjutkan
1
Masukkan Nilai yang akan di Push : 2_

```

### ***5.3. Double Stack dengan Array***

Metode ini adalah teknik khusus yang dikembangkan untuk menghemat pemakaian memori dalam pembuatan dua stack dengan array. Intinya adalah penggunaan hanya sebuah array untuk menampung dua stack.

Tampak jelas bahwa sebuah array dapat dibagi untuk dua stack, stack 1 bergerak ke atas dan stack 2 bergerak ke bawah. Jika Top1 (elemen teratas dari Stack 1) bertemu dengan Top 2 (elemen teratas dari Stack 2) maka double stack telah penuh.

Implementasi double stack dengan array adalah dengan memanfaatkan operasi-operasi yang tidak berbeda jauh dengan operasi single stack dengan array.

#### **Operasi-operasi Double Stack Array**

##### ***IsFull***

Fungsi ini memeriksa apakah double stack sudah penuh. Stack dianggap penuh jika Top[0] dan Top[1] bersentuhan sehingga stack tidak memiliki ruang kosong. Dengan kata lain,  $(\text{Top}[0] + 1) \geq \text{Top}[1]$ .

##### ***Push***

Fungsi ini memasukkan sebuah elemen ke salah satu stack.

##### ***IsEmpty***

Fungsi memeriksa apakah stack pertama atau stack kedua kosong. Stack pertama dianggap kosong jika puncak stack bernilai kurang dari nol, sedangkan stack kedua dianggap kosong jika puncak stack sama atau melebihi MAX\_STACK.

##### ***Pop***

Fungsi ini mengeluarkan elemen teratas dari salah satu stack

### ***Clear***

Fungsi ini mengosongkan salah satu stack.

## ***5.4. Stack dengan Single Linked List***

Selain implementasi stack dengan array seperti telah dijelaskan sebelumnya, ada cara lain untuk mengimplementasi stack dalam C++, yakni dengan single linked list. Keunggulannya dibandingkan array tebtu saja adalah penggunaan alokasi memori yang dinamis sehingga menghindari pemborosan memori. Misalnya saja pada stack dengan array disediakan tempat untuk stack berisi 150 elemen, sementara ketika dipakai oleh user stack hanya diisi 50 elemen, maka telah terjadi pemborosan memori untuk sisa 100 elemen, yang tak terpakai. Dengan penggunaan linked list maka tempat yang disediakan akan sesuai dengan banyaknya elemen yang mengisi stack. Oleh karena itu pula dalam stack dengan linked list tidak ada istilah full, sebab biasanya program tidak menentukan jumlah elemen stack yang mungkin ada (kecuali jika sudah dibatasi oleh pembuatnya). Namun demikian sebenarnya stack ini pun memiliki batas kapasitas, yakni dibatasi oleh jumlah memori yang tersedia.

### **Operasi-operasi untuk Stack dengan Linked List**

#### ***IsEmpty***

Fungsi memeriksa apakah stack yang adamasih kosong.

#### ***Push***

Fungsi memasukkan elemen baru ke dalam stack. Push di sini mirip dengan insert dalam single linked list biasa.

#### ***Pop***

Fungsi ini mengeluarkan elemen teratas dari stack.

## ***Clear***

Fungsi ini akan menghapus stack yang ada.

### **Contoh Program:**

#### **1. Stack dengan Single Linked List**

```
#include <iostream.h>
#include <stdlib.h>

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

struct node *push(struct node *p, int nilai)
{
    struct node *temp;
    temp = (struct node *) malloc(sizeof(struct node));

    /*Membuat Node Baru, menggunakan data nilai
    sebagai parameter*/
    if (temp == NULL)
    {
        cout<<"Error !!!";
        exit(0);
    }

    temp -> data = nilai;
    temp -> link = p;
    p = temp;
    return (p);
}

struct node *pop(struct node *p, int *nilai)
{
    struct node *temp;

    if (p == NULL)
    {
        cout<<"POP tidak dapat dilakukan, stack kosong"<<endl;
        exit(0);
    }
}
```

```

        *nilai = p -> data;
        temp = p;
        p = p -> link;
        free(temp);
        return(p);
    }

void main()
{
    struct node *top = NULL;
    int n, nilai;

    do
    {
        do
        {
            cout<<"Insert Nilai Elemen, PUSH : ";
            cin>>nilai;
            top = push(top, nilai);
            cout<<"Tekan 1 untuk Melanjutkan : ";
            cin>>n;
        } while (n == 1);

        cout<<"Tekan 1 untuk POP Elemen : ";
        cin>>n;
        cout<<endl;

        while (n == 1)
        {
            top = pop(top, &nilai);
            cout<<"Nilai yang di POP : "<<nilai;
            cout<<endl;
            cout<<"Tekan 1 untuk POP Elemen : ";
            cin>>n;
        }

        cout<<endl;
        cout<<"Enter 1 untuk Melanjutkan : ";
        cin>>n;
    } while (n == 1);
}

```

Output:

```
Insert Nilai Elemen, PUSH : 12
Tekan 1 untuk Melanjutkan : 1
Insert Nilai Elemen, PUSH : 124
Tekan 1 untuk Melanjutkan : 1
Insert Nilai Elemen, PUSH : 25
Tekan 1 untuk Melanjutkan : 2
Tekan 1 untuk POP Elemen : 1
```

```
Nilai yang di POP : 25
Tekan 1 untuk POP Elemen : 1
Nilai yang di POP : 124
Tekan 1 untuk POP Elemen : 1
Nilai yang di POP : 12
Tekan 1 untuk POP Elemen : 3
```

```
Enter 1 untuk Melanjutkan : 1
Insert Nilai Elemen, PUSH : 12
Tekan 1 untuk Melanjutkan : 2
Tekan 1 untuk POP Elemen : 2
```

```
Enter 1 untuk Melanjutkan : 2
```

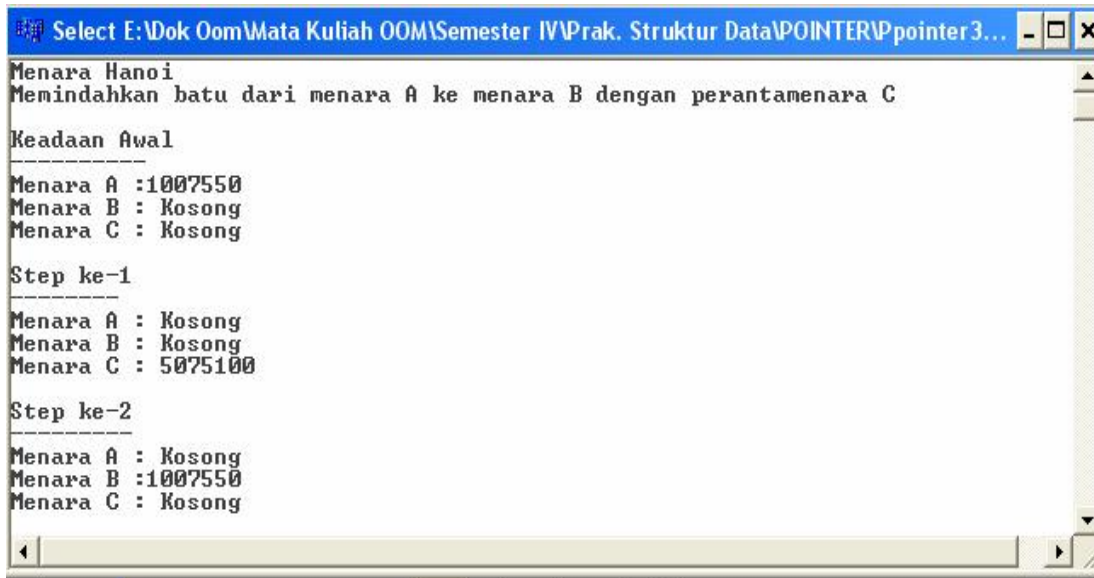
Latihan: Kasus Menara Hanoi – Menggunakan Turbo C++ 4.5

Memindahkan lempengan dari menara A ke menara B dengan perantara menara C

dengan jumlah data = 3 (50, 75, 100). Step program:

1. Pindahkan batu 50 dari A ke C
2. Pindahkan batu 75 dari A ke C
3. Pindahkan batu 100 dari A ke B
4. Pindahkan batu 75 dari C ke B
5. Pindahkan batu 50 dari C ke B

## Output:



```
Select E:\Dok Oom\Mata Kuliah OOM\Semester IV\Prak. Struktur Data\POINTER\ppointer3...
Menara Hanoi
Memindahkan batu dari menara A ke menara B dengan perantamenara C

Keadaan Awal
-----
Menara A :1007550
Menara B : Kosong
Menara C : Kosong

Step ke-1
-----
Menara A : Kosong
Menara B : Kosong
Menara C : 5075100

Step ke-2
-----
Menara A : Kosong
Menara B :1007550
Menara C : Kosong
```



## Bab 6. QUEUE

### 6.1. Definisi Queue

Jika diartikan secara harafiah, queue berarti antrian, queue merupakan salah satu contoh aplikasi dari pembuatan double linked list yang cukup sering kita temui dalam kehidupan sehari-hari, misalnya saat Anda mengantri di loket untuk membeli tiket.

Istilah yang cukup sering dipakai seseorang masuk dalam sebuah antrian adalah enqueue. Dalam suatu antrian, yang datang terlebih dahulu akan dilayani lebih dahulu.

Istilah yang sering dipakai bila seseorang keluar dari antrian adalah dequeue.

Walaupun berbeda implementasi, struktur data queue setidaknya harus memiliki operasi-operasi sebagai berikut :

EnQueue	Memasukkan data ke dalam antrian
DeQueue	Mengeluarkan data terdepan dari antrian
Clear	Menghapus seluruh antrian
IsEmpty	Memeriksa apakah antrian kosong
IsFull	Memeriksa apakah antrian penuh

### 6.2. Implementasi Queue dengan Linear Array

#### Linear Array

Linear array adalah suatu array yang dibuat seakan-akan merupakan suatu garis lurus dengan satu pintu masuk dan satu pintu keluar.

Berikut ini diberikan deklarasi kelas Queue Linear sebagai implementasi dari Queue menggunakan linear array. Dalam prakteknya, anda dapat menggantinya sesuai dengan kebutuhan Anda. Data diakses dengan field data, sedangkan indeks item pertama dan terakhir disimpan dalam field Head dan Tail. Konstruktor akan menginisialisasikan nilai Head dan Tail dengan -1 untuk menunjukkan bahwa antrian masih kosong dan

mengalokasikan data sebanyak MAX\_QUEUE yang ditunjuk oleh Data. Destruktor akan mengosongkan antrian kembali dan mendealokasikan memori yang digunakan oleh antrian.

### Operasi-Operasi Queue dengan Linear Array

#### *IsEmpty*

Fungsi IsEmpty berguna untuk mengecek apakah queue masih kosong atau sudah berisi data. hal ini dilakukan dengan mengecek apakah tail bernilai -1 atau tidak. Nilai -1 menandakan bahwa queue masih kosong.

#### *IsFull*

Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bisa menampung data dengan cara mengecek apakah nilai tail sudah sama dengan jumlah maksimal queue. Jika nilai keduanya sama, berarti queue sudah penuh.

#### *EnQueue*

Fungsi EnQueue berguna untuk memasukkan sebuah elemen dalam queue.

#### *DeQueue*

Fungsi DeQueue berguna untuk mengambil sebuah elemen dari queue. Operasi ini sering disebut juga serve. Hal ini dilakukan dengan cara memindahkan sejauh satu langkah ke posisi di depannya sehingga otomatis elemen yang paling depan akan tertimpa dengan elemen yang terletak di belakangnya.

#### *Clear*

Fungsi Clear berguna untuk menghapus semua lemen dalam queue dengan jalan mengeluarkan semua elemen tersebut satu per satu hingga queue kosong dengan memanfaatkan fungsi DEQueue.

### ***6.3. Implementasi Queue dengan Circular Array***

#### **Circular Array**

Circular array adalah suatu array yang dibuat seakan-akan merupakan sebuah lingkaran dengan titik awal (head) dan titik akhir (tail) saling bersebelahan jika array tersebut masih kosong.

Posisi head dan tail pada gambar diatas adalah bebas asalkan saling bersebelahan. Berikut ini diberikan deklarasi kelas Queue Circular sebagai implementasi circular array. Dalam prakteknya, Anda dapat menggantikanny sesuai dengan kebutuhan Anda. Data diakses dengan field data, sedangkan indeks itemn pertama dan terakhir disimpan dalam field Head dan Tail. Konstruktor akan menginisialisasi nilai Head dan Tail dengan 0 dan MAX-QUEUE-1 untuk menunjukkan bahwa antrian masih kosong dan mengalokasikan data sebanyak MAX-QUEUE yang ditunjuk oleh Data. destruktur akan mengosongkan antrian kembali dan mendealokasikan memori yang digunakan oleh antrian.

#### **Operasi-Operasi Queue dengan Circular Array**

##### ***IsEmpty***

Fungsi IsEmpty berguna untuk mengecek apakah Queue masih kosong atau sudah berisi. Hal ini dilakukan dengan mengecek apakah tail masih terletak bersebelahan dengan head dan tail lebih besar dari head atau tidak. Jika benar, maka queue masih kosong.

##### ***IsFull***

Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bias menampung data dengan cara mengecek apakah tempat yang masih kosong tinggal

satu atau tidak (untuk membedakan dengan empty dimana semua tempat kosong). Jika benar berarti queue penuh.

### ***EnQueue***

Fungsi EnQueue berguna untuk memasukkan sebuah elemen ke dalam queue tail dan head mula-mula bernilai nol (0).

### ***DeQueue***

DeQueue berguna untuk mengambil sebuah elemen dari queue. Hal ini dilakukan dengan cara memindahkan posisi head satu langkah ke belakang.

## ***6.4. Implementasi Queue dengan Double Linked List***

Selain menggunakan array, queue juga dapat dibuat dengan linked list. Metode linked list yang digunakan adalah double linked list.

### **Operasi-operasi Queue dengan Double Linked List**

#### ***IsEmpty***

Fungsi IsEmpty berguna untuk mengecek apakah queue masih kosong atau sudah berisi data. Hal ini dilakukan dengan mengecek apakah head masih menunjukkan pada Null atau tidak. Jika benar berarti queue masih kosong.

#### ***IsFull***

Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bias menampung data dengan cara mengecek apakah Jumlah Queue sudah sama dengan MAX\_QUEUE atau belum. Jika benar maka queue sudah penuh.

### ***EnQueue***

Fungsi EnQueue berguna untuk memasukkan sebuah elemen ke dalam queue (head dan tail mula-mula meunjukkan ke NULL).

### ***DeQueue***

Procedure DeQueue berguna untuk mengambil sebuah elemen dari queue. Hal ini dilakukan dengan cara menghapus satu simpul yang terletak paling depan (head).

### **Contoh Program :**

#### **1. Queue dengan Menggunakan Array**

```
#include <iostream.h>
#include <stdlib.h>

#define MAX 10 //Ukuran Maksimum Queue

void insert (int queue[], int *rear, int nilai);
void del (int queue[], int *front, int rear, int *nilai);

void main()
{
    int queue[MAX];
    int front, rear;
    int n, nilai;

    front = rear = (-1);
    do
    {
        do
        {
            cout<<"Masukkan Nilai Elemen : ";
            cin>>nilai;
            insert (queue,&rear,nilai);

            cout<<endl;
            cout<<"Tekan 1 untuk Melanjutkan : ";
            cin>>n;
        } while (n == 1);

        cout<<endl;
        cout<<"Tekan 1 untuk Menghapus Sebuah Elemen : ";
        cin>>n;
```

```

while (n == 1)
{
    del(queue,&front,rear,&nilai);
    cout<<"Nilai telah Dihapus : "<<nilai<<endl;
    cout<<endl;
    cout<<"Tekan 1 untuk Menghapus Sebuah Elemen : ";
    cin>>n;
}

cout<<endl;
cout<<"Tekan 1 untuk Melanjutkan : ";
cin>>n;
} while (n == 1);
}

void insert (int queue[], int *rear, int nilai)
{
    if (*rear < MAX-1)
    {
        *rear = *rear + 1;
        queue[*rear] = nilai;
    }
    else
    {
        cout<<"Queue Penuh, Insert Tidak Dapat Dilakukan"<<endl;
        exit(0);
    }
}

void del (int queue[], int *front, int rear, int *nilai)
{
    if (*front == rear)
    {
        cout<<"Queue Kosong, Delete Tidak Dapat Dilakukan"<<endl;
        exit(0);
    }

    *front = *front + 1;
    *nilai = queue[*front];
}

```

Output:

```
Masukkan Nilai Elemen : 78
Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 85
Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 78
Tekan 1 untuk Melanjutkan : 2
Tekan 1 untuk Menghapus Sebuah Elemen : 1
Nilai telah Dihapus : 78
Tekan 1 untuk Menghapus Sebuah Elemen : 1
Nilai telah Dihapus : 85
Tekan 1 untuk Menghapus Sebuah Elemen : 2
Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 45
Tekan 1 untuk Melanjutkan : 2
Tekan 1 untuk Menghapus Sebuah Elemen : 2
```

## 2. Queue menggunakan Linked List

```
#include <iostream.h>
#include <stdlib.h>

#define Nil NULL

struct node
{
    int data;
    struct node *link;
};
```

```

void insert (struct node **front, struct node **rear, int nilai)
{
    struct node *temp;

    temp = (struct node *)malloc(sizeof(struct node));

    /*Buat Node Baru dengan Menggunakan Nilai Data sebagai
       parameter */

    if (temp == Nil)
    {
        cout<<"Error, Memori Penuh"<<endl;
        exit(0);
    }

    temp -> data = nilai;
    temp -> link = Nil;

    if (*rear == Nil)
    {
        *rear = temp;
        *front = *rear;
    }

    else
    {
        (*rear) -> link = temp;
        *rear = temp;
    }
}

void del(struct node **front, struct node **rear, int *nilai)
{
    struct node *temp;

    if ((*front == *rear) && (*rear == Nil))
    {
        cout<<"Queue Kosong, Delete Tidak Dapat Dilakukan"<<endl;
        exit(0);
    }

    *nilai = (*front) -> data;
    temp = *front;
    *front = (*front) -> link;

    if(*rear == temp)
    *rear = (*rear) -> link;
    free(temp);
}

```



```

void main()
{
    struct node *front = Nil, *rear = Nil;
    int n,nilai;

    do
    {
        do
        {
            cout<<"Masukkan Nilai Elemen : ";
            cin>>nilai;
            cout<<endl;
            insert(&front,&rear,nilai);
            cout<<"Tekan 1 untuk Melanjutkan : ";
            cin>>n;
        } while (n == 1);

        cout<<endl;
        cout<<"Tekan 1 untuk Menghapus Elemen : ";
        cin>>n;

        while (n == 1)
        {
            del(&front,&rear,&nilai);
            cout<<endl;
            cout<<"Nilai yang Dihapus : "<<nilai<<endl;
            cout<<"Tekan 1 untuk Menghapus Elemen : ";
            cin>>n;
        }

        cout<<endl;
        cout<<"Tekan 1 untuk Melanjutkan : ";
        cin>>n;
    } while (n == 1);
}

```

Output:

```
Masukkan Nilai Elemen : 7
Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 12
Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 35
Tekan 1 untuk Melanjutkan : 2
Tekan 1 untuk Menghapus Elemen : 1
Nilai yang Dihapus : 7
Tekan 1 untuk Menghapus Elemen : 1
Nilai yang Dihapus : 12
Tekan 1 untuk Menghapus Elemen : 2
Tekan 1 untuk Melanjutkan : 2
```

## **Bab 7. TREE**

### **7.1. Definisi Tree**

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hierarkis (hubungan one to many) antara elemen-elemen. Tree bias didefinisikan sebagai kumpulan simpul/node dengan elemen khusus yang disebut Root. Node lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lain (disebut Subtree). Untuk lebih jelasnya, di bawah akan diuraikan istilah-istilah umum dalam tree.

<b>Predecessor</b>	Node yang berada di atas node tertentu
<b>Successor</b>	Node yang berada dibawah node tertentu
<b>Ancestor</b>	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
<b>Descendant</b>	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
<b>Parent</b>	Predecessor satu level di atas suatu node
<b>Child</b>	Successor satu level di bawah suatu node
<b>Sibling</b>	Node-node yang memiliki parent yang sama dengan suatu node
<b>Subtree</b>	Bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
<b>Size</b>	Banyaknya node dalam suatu tree
<b>Height</b>	Banyaknya tingkatan / level dalam suatu tree
<b>Root</b>	Satu-satunya node khusus dalam tree yang tak punya predecessor
<b>Leaf</b>	Node-node dalam tree yang tak memiliki successor
<b>Degree</b>	Banyaknya child yang dimiliki suatu node

## **7.2. Jenis-Jenis Tree**

### ***Binary Tree***

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

Jenis- Jenis Binary Tree :

### ***Full Binary Tree***

Jenis binary tree ini tiap nodenya (kecuali leaf) memiliki dua child dan tiap subtree harus mempunyai panjang path yang sama.

### ***Complete Binary Tree***

Jenis ini mirip dengan Full Binary Tree, namun tiap subtree boleh memiliki panjang path yang berbeda dan setiap node kecuali leaf hanya boleh memiliki 2 child.

### ***Skewed Binary Tree***

Skewed Binary Tree adalah Binary Tree yang semua nodenya (kecuali leaf) hanya memiliki satu child.

### ***Implementasi Binary Tree***

Binary tree dapat diimplementasikan dalam C++ dengan menggunakan double linkedlist.

### 7.3. Operasi-Operasi pada Binary Tree

Create	Membentuk binary tree baru yang masih kosong
Clear	Mengosongkan binary tree yang sudah ada
Empty	Function untuk memeriksa apakah binary tree masih kosong
Insert	Memasukkan sebuah node ke dalam tree. Ada tiga pilihan insert : sebagai root, left child, atau right child. Khusus insert sebagai root, tree harus dalam keadaan kosong
Find	Mencari root, parent, left child, atau right child dari suatu node. (tree tidak boleh kosong).
Update	Mengubah isi dari node yang ditunjuk oleh pointer current (Tree tidak boleh kosong)
Retrieve	Mengetahui isi dari node yang ditunjuk oleh pointer current (Tree tidak boleh kosong)
DeleteSub	Menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk current. Tree tidak boleh kosong. Setelah itu, pointer current akan berpindah ke parent dari node yang dihapus.
Characteristic	Mengetahui karakteristik dari suatu tree, yakni: size, height, serta average length. Tree tidak boleh kosong.
Traverse	Mengunjungi seluruh node-node pada tree, masing-masing sekali. Hasilnya adalah urutan informasi secara linear yang tersimpan dalam tree. Ada tiga cara traverse,yaitu PreOrder, InOrder, dan PostOrder.

Langkah-langkah Tranverse :

- PreOrder : cetak isi node yang dikunjungi, kunjungi Left Child, kunjungi Right Child

- InOrder : kunjungi Left Child, cetak isi node yang dikunjungi, kunjungi Right Child
- PostOrder : kunjungi Left Child, kunjungi Right Child cetak isi node yang dikunjungi.

#### ***7.4. Binary Search Tree***

Binary Tree ini memiliki sifat dimana semua left child harus lebih kecil dari pada right child dan parentnya. Semua right child juga harus lebih besar dari left child serta parentnya. Binary search tree dibuat untuk mengatasi kelemahan pada binary tree biasa, yaitu kesulitan dalam searching / pendarian node tertentu dalam binary tree.

Pada dasarnya operasi dalam Binary Search Tree sama dengan Binary Tree biasa, kecuali pada operasi insert, update, dan delete.

##### ***Insert***

Pada Binary Search Tree insert dilakukan setelah lokasi yang tepat ditemukan (lokasi tidak ditentukan oleh user sendiri ).

##### ***Update***

Update ini seperti yang ada pada Binary Tree biasa, namun di sini update akan berpengaruh pada posisi node tersebut selanjutnya. Bila update mengakibatkan tree tersebut bukan Binary Search Tree lagi, harus dilakukan perubahan pada tree dengan melakukan rotasi supaya tetap menjadi Binary Search Tree.

##### ***Delete***

Seperti halnya update, delete dalam Binary Search Tree juga turut mempengaruhi struktur dari tree tersebut.

## AVL Tree

AVL Tree adalah Binary Search Tree yang memiliki perbedaan tinggi/ level maksimal 1 antara subtree kiri dan subtree kanan. AVL Tree muncul untuk menyeimbangkan Binary Search Tree. Dengan AVL Tree, waktu pencarian dan bentuk tree dapat dipersingkat dan disederhanakan.

Selain AVL Tree, terdapat pula Height Balanced n Tree, yakni Binary Search Tree yang memiliki perbedaan level antara subtree kiri dan subtree kanan maksimal adalah n sehingga dengan kata lain AVL Tree adalah Height Balanced 1 Tree.

Untuk memudahkan dalam menyeimbangkan tree, digunakan simbol-simbol Bantu :

- (tanda minus) : digunakan apabila Subtree kiri lebih panjang dari Subtree kanan.
- + (tanda plus) : digunakan apabila Subtree kanan lebih panjang dari Subtree kiri.
- 0 (nol) : digunakan apabila Subtree kiri dan Subtree kanan mempunyai height yang sama.

## DAFTAR ISTILAH-ISTILAH

Algoritma	: Langkah-langkah menyelesaikan suatu masalah yang disusun secara logis dan berurutan
Animasi	: Gambar yang tampak bergerak, terdiri dari banyak gambar-gambar tunggal (disebut frame) yang ditampilkan satu per satu secara bergantian dengan cepat sehingga objek dalam gambar tampak seolah-olah bergerak.
Array	: Struktur data yang memiliki banyak elemen di dalamnya, dengan masing-masing elemen memiliki tipe data yang sama.
Clear	: Menghapus secara keseluruhan, biasanya digunakan sebagai nama fungsi/metode yang bertujuan untuk mengosongkan list atau menghapus keseluruhan elemen.
Console	: Istilah dalam komputer yang menunjuk pada antarmuka

antara pemakai dengan komputer yang berbasis teks. Cara kerja konsol sangat sederhana yaitu menggunakan standar input untuk membaca input dari keyboard dan standar output untuk menampilkan teks ke layar monitor.

<b>Data</b>	: Informasi yang disimpan komputer, dapat berbentuk teks, gambar, suara, video, dan sebagainya.
<b>Delete</b>	: Menghapus sebuah elemen, biasanya digunakan sebagai nama fungsi/metode yang bertujuan untuk menghapus sebuah elemen dalam suatu list/tree
<b>Deret geometric</b>	: Deretan bilangan yang setiap bilangan merupakan hasil kali bilangan sebelumnya dengan suatu konstanta.
<b>Destruktor</b>	: Metode khusus dalam sebuah kelas untuk menghapus objek hasil instansiasi kelas tersebut
<b>Dimensi</b>	: Jumlah indek yang diperlukan untuk menyatakan sebuah elemen dalam array
<b>Elemen</b>	: Sebuah data tunggal yang paling kecil dari sebuah array atau list. Data tunggal disini tidak perlu data sederhana, melainkan bisa berupa kumpulan data atau list yang lain.
<b>Empty</b>	: Keadaan di mana list ada dalam keadaan kosong
<b>Fibonacci</b>	: Barisan bilangan yang setiap bilangan merupakan jumlah dari dua bilangan sebelumnya.
<b>Field</b>	: Data yang dimiliki oleh sebuah objek
<b>FIFO</b>	: First In First Out sifat suatu kumpulan data. jika sebuah elemen A dimasukkan lebih dulu dari B maka A harus dikeluarkan dulu dari B
<b>FPB</b>	: Faktor Persekutuan terbesar, faktor yang paling besar jika sejumlah bilangan memiliki beberapa faktor yang sama.
<b>Full</b>	: Keadaan di mana list penuh, tidak boleh menerima data lagi



<b>Fungsi</b>	: Suatu modul atau bagian program yang mengerjakan suatu program tertentu.
<b>Himpunan</b>	: Kumpulan dari objek-objek, misalnya sebuah himpunan dari buah-buahan dapat terdiri dari pisang, mangga, jambu dll.
<b>Indeks</b>	: Bilangan yang digunakan untuk menyatakan posisi suatu elemen dalam array atau list.
<b>Input</b>	: Data masukan, dalam fungsi berarti parameter yang dimasukkan, sedangkan dalam program secara keseluruhan berarti data yang dimasukkan pemakai, bias melalui parameter program, file maupun lewat keyboard
<b>Insert</b>	: Memasukkan sebuah elemen baru ke dalam list. Biasanya insert dilakukan baik di tengah-tengah list, awal, maupun di akhir list.
<b>Iterasi</b>	: Perulangan dengan struktur perulangan, while, do while, maupun for.
<b>Kelas</b>	: Suatu struktur yang digunakan sebagai template bagi objek-objek yang sama sifatnya.
<b>Kompilasi</b>	: Proses menerjemahkan bahasa sumber (source code) ke dalam bahasa lain, biasanya bahasa mesin, untuk dapat dijalankan langsung oleh computer melalui system operasi.
<b>Kompiler</b>	: Program yang mengerjakan kompilasi.
<b>Konstruktor</b>	: Metode khusus yang dimiliki suatu kelas untuk membentuk suatu objek baru berdasarkan kelas tersebut
<b>Library</b>	: Kumpulan fungsi, makro, template, dan kelas yang disediakan bersama compiler C++.
<b>LIFO</b>	: Last In First Out, sifat kumpulan data, kebalikan dari FIFO
<b>Linked List</b>	: List yang didesain dengan cara mendefinisikan sebuah elemen yang memiliki hubungan atau link dengan elemen lain yang

dihubungkan dengan elemen yang lain lagi.

<b>Matriks</b>	: Dalam matematika berarti kumpulan bilangan yang disusun dalam bentuk kolom dan baris.
<b>Metode</b>	: Fungsi yang dimiliki suatu objek
<b>Objek</b>	: Struktur data yang terdiri dari data yang lebih sederhana yang disebut field yang memiliki operasi sendiri untuk menangani data-data yang dimilikinya.
<b>Output</b>	: Data yang dihasilkan oleh program
<b>Pointer</b>	: Type data khusus yang pada umumnya berukuran 32 bit yang berfungsi untuk menampung bilangan tertentu yang menunjuk pada lokasi memory tertentu
<b>Pop</b>	: Mengeluarkan satu elemen dari dalam list dengan cara menyalin data elemen tersebut, kemudian menghapus elemen tersebut dari list biasanya digunakan untuk stack.
<b>Prima</b>	: Bilangan yang tidak memiliki faktor selain 1 dan bilangan itu sendiri.
<b>Push</b>	: Memasukkan sebuah elemen baru ke dalam list.
<b>Queue</b>	: Struktur list dengan sifat FIFO, cara kerjanya seperti antrian manusia.
<b>Record</b>	: Struktur data yang terdiri dari satu atau lebih elemen yang tipe data bias berbeda.
<b>Rekursi</b>	: Jenis perulangan yang tidak menggunakan struktur perulangan, tetapi dengan memanggil fungsi yang bersangkutan.
<b>Sort</b>	: Menyusun elemen-elemen suatu list secara berurutan.
<b>Source Code</b>	: Program yang ditulis menggunakan bahasa pemrograman tertentu. Kode sumber belum dapat dijalankan oleh komputer dan perlu menjalani proses kompilasi sehingga dapat

dijalankan.

- Stack** : List yang memiliki sifar LIFO. Data yang hendak di dikeluarkan dari stack haruslah data yang paling terakhir dari stack.
- STL** : Standar Template Library merupakan kumpulan yang disertakan dalam setiap compiler C++ yang memenuhi standar ANSI C++ yang menyediakan berbagai struktur data, algoritma dan template yang sering dipakai.
- Stream** : Aliran merupakan konsep dalam C++ untuk input dan ouput data tanpa memperdulikan isi data maupun media penampung data tersebut.
- Struktur kontrol** : Struktur yang digunakan untuk mengontrol jalannya program.
- Teks** : Data yang terdiri dari karakter-karakter yang dapat dibaca (huruf bilangan, tanda baca).
- Tree** : Suatu struktur data yang setiap elemen terhubung sedemikian rupa sehingga berbentuk seperti pohon.

#### Contoh Program:

1.

```
#include <iostream.h>
#include <malloc.h>

#define Nil NULL

struct nod
{
    struct nod *left;
    char data;
    struct nod *right;
};

typedef struct nod NOD;
typedef NOD POKOK;
```

```

NOD *NodBaru(char item)
{
    NOD *n;

    n = (NOD *)malloc(sizeof(NOD));

    if (n != Nil)
    {
        n -> data = item;
        n -> left = Nil;
        n -> right = Nil;
    }

    return n;
}

void BinaPokok (POKOK **T)
{
    *T = Nil;
}

typedef enum {FALSE = 0, TRUE = 1} BOOL;

BOOL PokokKosong (POKOK *T)
{
    return ((BOOL) (T == Nil));
}

void TambahNod(NOD **p, char item)
{
    NOD *n;

    n = NodBaru(item);

    *p = n;
}

void preOrder(POKOK *T)
{
    if (!PokokKosong(T))
    {
        cout<<" "<< T -> data;
        preOrder(T -> left);
        preOrder(T -> right);
    }
}

```

```

void inOrder (POKOK *T)
{
    if (!PokokKosong(T))
    {
        inOrder(T -> left);
        cout<<" "<< T -> data;
        inOrder(T -> right);
    }
}

void postOrder (POKOK *T)
{
    if (!PokokKosong(T))
    {
        postOrder(T -> left);
        postOrder(T -> right);
        cout<<" "<< T -> data;
    }
}

int main()
{
    POKOK *kelapa;
    char buah;

    BinaPokok(&kelapa);

    TambahNod(&kelapa, buah = 'M');

    TambahNod(&kelapa -> left, buah = 'E');

    TambahNod(&kelapa -> left -> right, buah = 'I');

    TambahNod(&kelapa -> right, buah = 'L');

    TambahNod(&kelapa -> right -> right, buah = 'O');

    TambahNod(&kelapa -> right -> right -> left, buah = 'D');
}

```

```

cout<<"Tampilan secara PreOrder : ";
preOrder(kelapa);

cout<<endl;
cout<<"Tampilan secara InOrder : ";
inOrder(kelapa);

cout<<endl;
cout<<"Tampilan secara PostOrder : ";
postOrder(kelapa);

cout<<endl;
cout<<endl;

return 0;
}

```

Output:

```

Tampilan secara PreOrder : M E I L O D
Tampilan secara InOrder : E I M L D O
Tampilan secara PostOrder : I E D O L M

```

## REFERENSI

- Desphande P.S., O.G. Kakde (2004). *C dan Data Structures*. Charles River Media, Inc. Massachusetts
- Heriyanto, Imam, Budi Raharjo (2003). *Pemrograman Borland C++ Builder*. Informatika Bandung.
- Indrajit, Richardus Eko. *Manajemen Sistem Informasi dan Teknologi Informasi*.
- Indrajit, Richardus Eko. *Kajian Strategis Analisa Cost-Benefit Investasi Teknologi Informasi*.
- Lidya, Leoni, rinaldi Munir (2002). *Algoritama dan Pemrograman dalam Bahas Pascal dan C*. Informatika Bandung.
- Sanjaya, Dwi (2005). *Asyiknya Belajar Struktur Data di Planet C++*. Elex Media Komputindo.
- Solichin, Achmad (2003). *Pemrograman Bahasa C dengan Turbo C*. IlmuKomputer.Com.
- Wahono, Romi Satria(2003). *Cepat MahirBahasa*. IlmuKomputer.Com.