


2019/2020

STRUKTUR DATA DAN ALGORITMA LANJUT



	LEMBAR PENGESAHAN BUKU AJAR/BAHAN AJAR/MODUL/DIKTAT		Nomor Dok : FRM/PMB/D1/11
			Nomor Revisi : (0)
			Tgl. Berlaku : 1 Agustus 2016
			Klaus ISO : 7.6

Buku Ajar/Bahan Ajar / Modul / Diktat ini disusun oleh:

Nama Ketua : Siti Sa'uda, M.Kom

Anggota : 1.

2.

3. dst

Dan digunakan sebagai Bahan Ajar/ Modul/ Diktat pada:

Mata Kuliah : Struktur Data dan Algoritma Lanjutan

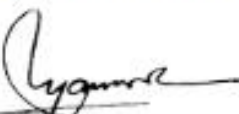
Semester/Tahun Akademik : Genap / 2019-2020

Fakultas/ Program Studi : Ilmu Komputer / Teknik Informatika

Universitas : Universitas Bina Darma

Disahkan pada Tanggal : Februari 2020

Fakultas Ilmu Komputer
Dekan,
Universitas Bina Darma



Dedy Syamsuar, Ph.D.

Program Studi Teknik Informatika
Ketua,
Universitas Bina Darma



Alek Wijaya, S.Kom., M.I.T.

KATA PENGANTAR

Puji syukur penulis panjatkan ke Hadirat Allah SWT yang telah melimpahkan rahmat, karunia, serta hidayah-Nya sehingga penyusunan Diktat ini sebagai bahan ajar yang mempunyai judul "Struktur Data" dapat penulis selesaikan.

Terimakasih penulis ucapkan kepada semua pihak yang telah membantu dalam penulisan Diktat Struktur Data ini. Penulis mengharapkan saran dan kritik yang membangun demi kesempurnaan Diktat ini. Semoga Diktat ini dapat bermanfaat khususnya bagi para pembaca.

Palembang, Februari 2020

Penyunting

DAFTAR ISI

BAB I ARRAY	1
BAB II DASAR STRING	13
BAB III POINTER	24
BAB IV STRUKTUR	38
BAB V SORT	55
BAB VI LINKED LIST	66
BAB VII STACK (TUMPUKAN)	80
BAB VIII QUEUE (ANTRIAN)	94
BAB IX TREE (POHON)	109

BAB I

ARRAY

Array adalah kumpulan data bertipe sama yang menggunakan nama sama.

Dengan menggunakan array, sejumlah variabel dapat memakai nama yang sama. Antara satu variabel dengan variabel lain di dalam array dibedakan berdasarkan *subscript*. Sebuah *subscript* berupa bilangan di dalam kurung siku.

Array dapat dibedakan menjadi :

1. Array berdimensi satu
2. Array berdimensi dua
3. Array berdimensi tiga

Array Berdimensi Satu

Contoh array berdimensi satu, misalnya menginputkan 5 buah data temperatur. Dan kelima data tersebut disimpan pada array bernama suhu.

Contoh 1 :

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

```
void main()
{
```

```
    float suhu[5];          // array dengan 5 elemen bertipe float
    // Membaca data dari keyboard dan meletakkan
    ke array cout << "Masukkan 5 buah data suhu" <<
    endl;
```

```
for (int i=0; i<5; i++)
{
    cout << i + 1 << " : ";
    cin >> suhu[i];
}
// Menampilkan isi array ke layar
cout << "Data suhu yang dimasukkan : " << endl;
for (i=0; i<5; i++)
    cout << suhu[i]
        << endl;
}
```

Mendefinisikan array

Float suhu[5];

float : Tipe elemen array

suhu : Nama array

[5] : Jumlah elemen array

Maka array **suhu** dapat menyimpan data sebanyak 5 buah.

Subscript dari array **selalu dimulai dari nol**. Misal, jika jumlah elemen array [5], maka index dari array tersebut yaitu 0, 1, 2, 3, 4.

Mengakses elemen array

Setelah suatu array didefinisikan, elemen array dapat diakses dengan bentuk :

<i>Nama_array[subscript]</i>

suhu[i] menyatakan “elemen suhu dengan subscript sama dengan i”

Perintah seperti `cin >> suhu[i];` berarti “membaca data dari keyboard dan meletakkan ke elemen nomor i pada array suhu”.
Perintah seperti `cout >> suhu[i];` berarti “menampilkan elemen bernomor i pada array suhu”.

Contoh 2 :

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

const int jum_data = 5;
void main()
{
    float suhu[jum_data];           // array suhu
    float total;                    // untuk menampung total
    suhu

    // Membaca data dari keyboard dan meletakkan
    ke array cout << "Masukkan 5 buah data suhu" <<
    endl;
    for (int i=0; i<5; i++)
    {
        cout << i + 1 << " : ";
        cin >> suhu[i];
    }
    // Menghitung nilai rata-rata
    total = 0;                      // Mula-mula diisi
    dengan nol for(i=0; i<jum_data; i++)
        total += suhu[i];          // Tambahkan isi suhu[i] ke
        total

    cout << "Suhu rata-rata= " << total/jum_data << endl;
}
```

Memberikan nilai awal terhadap array

Seperti halnya variabel biasa, array juga dapat diberi nilai awal

(diinisialisasikan) pada saat didefinisikan. Misalnya:

```
int jum_hari[12]={ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
                  30, 31};
```

Catatan:

C++ secara otomatis akan memberikan nilai awal nol terhadap array yang **bersifat global**. Jika **bersifat lokal**, maka harus diatur terlebih dahulu.

Contoh 3 :

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>

void main()
{
    // Pendefinisian array jum_hari dan pemberian nilai awal
    int jum_hari[12]={ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
    // Tampilkan isi jum_hari
    cout << "Masukkan 5 buah data suhu" << endl;
    for (int i=0; i<12; i++)
    {
        cout << "jum_hari[" << i << "] ="
              << jum_hari[i] << endl;
    }
}
```


Array Berdimensi Dua

Sebagai gambaran, data kelulusan dari jurusan Teknik Informatika, Manajemen Informatika, dan Teknik Komputer dari tahun 1992 hingga 1995.

Nama	1992	1993	1994	1995
Teknik Informatika	35	45	80	120
Manajemen Informatika	100	110	70	101
Teknik Komputer	10	15	20	17

Mendefinisikan array berdimensi dua

Bentuk diatas dapat dibentuk dalam array berdimensi dua, pendefinisiannya :

```
int nilai[3][4];
```

Pada pendefinisian di atas :

- 3 menyatakan jumlah baris (mewakili nama)
- 4 menyatakan jumlah kolom (mewakili nilai)

Mengakses array berdimensi dua

Masing-masing elemen di dalam array berdimensi dua dapat diakses dengan bentuk :

```
nama_array[subscript_baris, subscript_kolom]
```

Baris dan kolom dimulai dari 0.

Contoh pengaksesan elemen array berdimensi dua :

1. `data_lulus[1][2] = 5;`
Merupakan instruksi untuk memberikan nilai 5 ke baris 1 kolom 2.
2. `cout << data_lulus[1][2];`

Merupakan perintah untuk menampilkan elemen `data_lulus` dengan *subscript* pertama (baris) berupa 1 dan *subscript* kedua (kolom) bernilai 2.

Contoh 4 :

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

```
#include<conio.h>
```

```
void main()
```

```
{
```

```
    int data_lulus[3][4];           // Array  
    berdimensi dua int tahun, jurusan;
```

```
    // Memberikan data ke elemen array
```

```
    data_lulus[0][0] = 35;
```

```
    // TI - 1992 data_lulus[0][1] = 45;
```

```
    // TI - 1993 data_lulus[0][2] = 90;
```

```
    // TI - 1994 data_lulus[0][3] = 120;
```

```
    // TI - 1995 data_lulus[1][0] = 100;
```

```
    // MI - 1992 data_lulus[1][1] = 110;
```

```
    // MI - 1993 data_lulus[1][2] = 70;
```

```
    // MI - 1994 data_lulus[1][3] = 101;
```

```
    // MI - 1995 data_lulus[2][0] = 10;
```

```
    // TK - 1992 data_lulus[2][1] = 15;
```

```
    // TK - 1993 data_lulus[2][2] = 20;
```

```
    // TK - 1994 data_lulus[2][3] = 17;
```

```
    // TK - 1995
```

```
    // Proses untuk memperoleh informasi  
    kelulusan while(1)
```

```
{
    cout << "Jurusan (0 = TI, 1 = MI, 2 = TK): ";
    cin >> jurusan;
    if ((jurusan==0) || (jurusan==1) || (jurusan==2))
        break; // keluar dari while
}
while(1)
{
    cout << "Tahun (1992 - 1995): ";
    cin >> tahun;

    if ((tahun >= 1992) && (tahun <= 1995))
    {
        tahun -= 1992; // konversi ke 0, 1,
        // 2 atau 3 break;
        keluar dari while
    }
}
cout << "Jumlah yang lulus = "
    << data_lulus[jurusan][tahun] << endl;
}
```

Melewatkan Array Sebagai Argumen Fungsi

Array juga dapat berkedudukan sebagai parameter di dalam fungsi.

Misalnya :

```
const int MAKS = 5 int data[MAKS];
```

Dari data di atas, fungsi yang menerima array di atas dapat dibuat prototipe-nya sebagai berikut :

```
Void inisialisasi_data(data[MAKS]);
```

Dan deklarasi fungsi sebagai berikut :

```
void inisialisasi_data(data[], int & jumlah);
```

Pada contoh kedua, tanda di dalam tanda [] tidak terdapat apa-apa dan parameter kedua digunakan untuk menyatakan jumlah elemen array

serta berkedudukan sebagai referensi (bisa diubah dari dalam fungsi `inisialisasi_data()`).

Contoh 5 :

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

const int MAKS = 100;
void inisialisasi_data(int data[], int &jumlah);
void main()
{
    int data_acak[MAKS];           // array
    berdimensi satu int jumlah;
    inisialisasi_data(data_acak, jumlah);

    // Tampilkan elemen-
    elemen array cout << "Isi
    array : " << endl;
    for(int i=0; i<jumlah; i++)
        cout << data_acak[i] << endl;

// Definisi fungsi
void inisialisasi_data(int data[], int &jumlah)
{
    while(1)
    {
        cout << "Berapa jumlah data yang ingin"
        << endl; cout << "dibangkitkan secara
        acak (5 - 100) ? "; cin >> jumlah;

        if ((jumlah >= 5) && (jumlah<=100))
            break;
    }
    randomize();           // Menyetel pembangkit
    bilangan acak for(int i=0; i<jumlah; i++)
        data[i] = random(100);
}
```

Apabila array berdimensi dua hendak dilewatkan sebagai argumen fungsi, pendeklarasiannya dapat berupa semacam beriku t:

```
void isi_matriks(float mat[BARIS][KOLOM], int &brs, int &kol);
```

Atau cukup mengisi pada bagian kolom saja, seperti contoh dibawa

ini :

```
void isi_matriks(float mat[][KOLOM], int &brs, int &kol);
```

Contoh 6 :

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>

const int BARIS = 5;
const int KOLOM = 5;
void isi_matriks(float mat[][KOLOM], int &brs, int &kol);

void main()
{
    float
    matriks[BARIS][KOLOM];
    int    jum_baris,
    jum_kolom;
    int i, j;
    isi_matriks(matriks,    jum_baris,
    jum_kolom); cout << "\nMatriks
    yang terbentuk: " << endl; cout <<
    setiosflags(ios::fixed);
    // ios::fixed --> mengatur angka yg dimasukkan dalam bentuk
    angka biasa
    // bukan dalam bentuk
    eksponen for (i=0;
    i<jum_baris; i++)
    {
        for (j=0; j<jum_kolom; j++)
            cout << setw(12) << setprecision(5) << matriks[i][j];
            // setprecision(5) --> mengatur banyak angka
            dibelakang koma cout << endl;
        }
    }

    // Definisi Fungsi
    void isi_matriks(float mat[][KOLOM], int &brs, int &kol)
    {

        int i,j;

        cout << "Pastikan jumlah baris dan kolom" << endl;
        cout << "tidak melebihi 5" << endl;
```

```
cout << "Jumlah baris = ";
cin >> brs;
cout << "Jumlah kolom = ";
cin >> kol;
for (i=0; i< brs; i++)
    for (j=0; j<kol; j++)
    {
        cout << "Elemen " << i << ", " << j << " = ";
        cin >> mat[i][j];
    }
}
```

Mengurutkan Data

Salah satu mengurutkan data adalah dengan menggunakan *bubble sort*. Pengurutan dilakukan dengan membandingkan setiap elemen dengan seluruh elemen yang terletak sesudah posisinya.

Contoh 7 :

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
```

```
void main()
{
    int i, j, tmp, jumdata;
    int data[] = {5, 100, 20, 31, 77, 88, 99, 20, 55, 1};
    jumdata = sizeof(data)/sizeof(int);

    // Menampilkan data
    cout << "Data semula: " << endl;
    for(i=0; i<jumdata; i++)
        cout << setw(4) <<
            data[i];
    cout << endl;        // Pindah baris

    // Mengurutkan
    data for(i=0;
    i<jumdata-1;
    i++)
        for(j=i+1;
        j<jumdata; j++)
            if (data[i] > data[j])
            {
                tmp =
                data[i];
                data[i] =
                data[j];
                data[j]
                = tmp;
            }
}
```

```
    }  
  
    // Menampilkan data  
    cout << "Data setelah diurutkan: " << endl;  
    for (i=0; i<jumdata; i++)  
        cout << setw(4) <<  
            data[i];  
    cout << endl;        // Pindah baris  
}
```


BAB II

DASAR STRING

Pengantar String

String sangat memudahkan tugas pemogram. Dengan menggunakan string, pemogram dapat menampilkan pesan kesalahan, menampilkan prompt bagi masukan keyboard ataupun memberikan informasi pada layar dengan mudah. Seperti halnya tipe data yang lain, string dapat berupa konstanta atau variabel. Konstanta string sudah biasa anda sertakan pada program. Misalnya pada pernyataan :

```
cout << "C++"<<endl;
```

terdapat konstanta string "C++". Tetapi sejauh ini, variabel string belum diperkenalkan.

Konstanta String

Suatu konstanta string ditulis dengan awalan dan akhiran tanda petik ganda(" "), misalnya: "C++". Konstanta string seperti diatas disimpan dalam memori secara berurutan.

Setiap karakter menempati memory 1 byte. Setelah karakter yang terakhir terdapat karakter Null (karakter dengan nilai ASCII sama dengan nol atau disimbolkan dengan "\0", yaitu tanda \ diikuti dengan nol).

Bila suatu string hanya berisi karakter NULL, string disebut sebagai string kosong.

Variabel String

Variabel string adalah variabel yang dipakai untuk menyimpan string. Misalnnnya:

```
char teks[10];
```

merupakan pernyataan untuk mendefinisikan variabel string dengan panjang maksimal 9 karakter(sudah termasuk karakter NULL). Perlu

diketahui , pernyataan di atas tidak lain untuk mendefinisikan array bertipe karakter.

Memasukan Data String

Setelah suatu variabel string didefinisikan, Anda bisa mengisi data ke variabel tersebut. Pemasukan data dapat ditangani oleh data cin, seperti contoh program di bawah ini :

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

void main()
{
    char teks[13]; //string dengan panjang maksimal 12
    karakter clrscr(); //hapus layar

    cout<<"Masukan sebuah kata :"<<endl;
    cin>> teks;
    cout<< "Kata yang tercetak  :"<<teks;
}
```

Yang perlu diperhatikan adalah bahwa cin hanya dapat membaca sebuah kata. Artinya karakter-karakter yang terletak sesudah spasi tidak

bisa ditampilkan pada teks. Ini disebabkan operator << pada cin hanya bisa membaca masukan hingga terdapat spasi, tab atau enter.

Untuk menampilkan agar dapat terbaca solusinya adalah menambahkan fungsi **get()** pada objek cin (**cin.get()**) bisa dipakai untuk keperluan ini. Sebagai contoh seperti program dibawah ini :

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

void main()
{
    char teks[13]; //string dengan panjang maksimal 12
    karakter clrscr(); //hapus layar

    cout<<"Masukan sebuah kata :"<<endl;
    cin.get(teks, 13);
    cout<< "Kata yang tercetak  :"<<teks;
}
```

Tampak, bahwa karakter yan terletak sesudah spasi juga ikut disimpan pada teks. Sekarang kita lihat kalau data yang dimasukan lebih dari 13 karakter maka hanya 12 karakter pertama yang disimpan pada teks, mengingat argumen kedua dari fungsi **get()** diisi dengan 13 (satu karakter berisi Null).

Pada contoh program di atas dapat ditulis sebagai berikut :

```
cin.get(teks, 13)
```

Bisa juga ditulis sebagai

berikut :

```
cin. Get(teks, sizeof(teks));
```

Fungsi getline()

Suatu masalah akan timbul kalau `cin.get()` digunakan 2 kali seperti contoh program dibawah ini

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

void main()
{
    char
    nama[25];
    char
    alamat[35];
    clrscr();
    //hapus
    layar

    cout<<"Masukan Nama :"<<endl;
    cin.get(nama, sizeof(nama));
    cout<< "Masukan Alamat  :"<<endl;
    cin.get(alamat,sizeof(alamat));

    cout<<"Nama:"<<nama<<endl;
    cout<<"Alamat:"<<alamat<<endl;
}
```

Pada contoh di atas `cin.get()` pertama digunakan untuk membaca nama yang kedua untuk membaca alamat. Ternyata program tidak memberikan kesempatan untuk mengisi alamat. Hal ini terjadi karena `get()` yang pertama tidak membuang kode newline(`\n`). Oleh karena `get()` tidak mengabaikan spasi putih(spasi, tab , atau newline) maka `get()` kedua

menjadi tidak berfungsi sebagaimana mestinya. Cara untuk menyelesaikan masalah di atas dengan menggunakan fungsi **getline()**, karena fungsi ini dapat membuang sisa data yang tidak dibaca, termasuk newline itu sendiri. Pada contoh berikut, **get()** diganti dengan **getline()**.

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

void main()
{
    char
    nama[25];
    char
    alamat[35];
    clrscr();
    //hapus
    layar

    cout<<"Masukan Nama :"<<endl;
    cin.getline(nama, sizeof(nama)); cout<<
    "Masukan Alamat  :"<<endl;
    cin.getline(alamat,sizeof(alamat));

    cout<<"Nama:"<<nama<<endl;
    cout<<"Alamat:"<<alamat<<endl;
}
```

Tampak bahwa dengan menggunakan **getline()**, data alamat dapat diisi.

Fungsi strcpy

Bentuk dari **strcpy()** :

strcpy(string_target, string_awal)

Prototipe fungsi di atas ada pada file header **string.h**

Contoh program yang menggunakan **strcpy()**:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
    char teks[]="C++ Oke";
    char data [25];
    clrscr(); //hapus layar
    strcpy(data,teks); // mengcopy isi
    teks ke data cout<<"Isi data
    :"<<data<<endl;

}
```

Fungsi toupper dan tolower

Fungsi **toupper()** berguna untuk memperoleh huruf kapital dari suatu huruf kecil. Nilai balik dari fungsi ini akan berupa seperti argumennya kalau argumen tidak berisi huruf kecil.

Adapun fungsi **tolower()** adalah kebalikan dari **toupper()**. Fungsi ini memberikan nilai balik :

- ™ Berupa huruf kecil kalau argumen berisi huruf kapital
- ™ Berupa nilai seperti argumen kalau argumen tidak berupa huruf kecil

Kedua fungsi di atas memberikan nilai balik bertipe **int** dan memiliki protipe pada file **ctype**.

Contoh program :

```
#include<iostream.h>
#include<conio.h>
#include<ctype.h>
void main()
{
    char st[]="saya
    suka C++";
    clrscr(); //hapus
```

```
        layar for(int i=0;
        st[i];i++)
        st[i]= toupper(st[i]);
        cout<<st<<endl;

    }
```

Pada contoh di atas,

st[i]= toupper(st[i]);
menyebabkan setiap huruf kecil pada variabel st akan diganti dengan huruf kapital.

Fungsi strlen()

Panjang suatu string dapat diketahui dengan mudah menggunakan fungsi **strlen()**. Misalnya saja, didefinisikan :

```
char bunga[15]= "mawar";
int panjang;
```

Maka pernyataan yang menggunakan strlen :

```
panjang = strlen(bunga);
```

akan memberikan panjang string yang tersimpan pada variabel bunga ke panjang.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
    char
    bunga[15]="maw
    ar"; char
    kosong[23]="
    clrscr(); //hapus
    layar
    cout<<strlen(bunga)<<endl;
    cout<<strlen(kosong)<<endl;

}
```

Fungsi `strlwr()` dan `strupr`

Jika isi semua huruf kapital pada suatu string diinginkan untuk diubah menjadi huruf kapital, hal ini dapat dilakukan melalui fungsi `strlwr()`. Misalnya, didefinisikan :

char

`st[]="AbCdEfGeHjKl";` Maka `st`

akan berisi : "abcdefgehjki";

Sedangkan fungsi `strupr()` kebalikan dari fungsi `strlwr()`. Kedua fungsi di

atas sama seperti `tolower` dan `toupper`.

Contoh program untuk memperlihatkan efek `strlwr()` dan `strupr()` :


```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
    char st[]="AbCdEfGhIjKl";
    clrscr(); //hapus layar
    cout<<"Isi St mula-mula:"<<st<<endl;
    strlwr(st);
    cout<<"Isi setelah dikonversi strlwr: "<<endl;
    cout<<st<<endl;
   strupr(st);
    cout<<"Isi setelah dikonversistrupr: "<<endl;
    cout<<st<<endl;
}
```

Konversi String ke Angka dan sebaliknya

Untuk melakukan konversi dari suatu string ke bilangan, dapat menggunakan sejumlah fungsi bawahan. Fungsi-fungsi yang tersedia dapat dilihat pada tabel.

Fungsi	Prototipe	Keterangan
atoi	Stdlib.h	Mengkonversi string argumen menjadi nilai bertipe int
atof	Stdlib.h	Mengkonversi string argumen menjadi nilai bertipe float
atol	Stdlib.h	Mengkonversi string argumen menjadi nilai bertipe long int
atold	Stdlib.h	Mengkonversi string argumen menjadi nilai bertipe long double

Beberapa contoh hasil pengonversian string ke bilangan :

```

f  atof("+2.1E+02")  = 210
f  atof("+2.")       = 2
f  atof("2abc")     = 2
f  atof(" 20.1")    = 20
f  atoi("+2.1E+02")  = 2
f  atoi("+2.")       = 2
f  atoi("2abc")      = 2
f  atoi(" 20.1")     = 20
f  atol("+2.1E+02")  = 2
f  atol("+2.")       = 2
f  atol("2abc")      = 2
f  atol(" 2000000000") = 2000000000
  
```

Adapun fungsi yang berguna untuk mengubah suatu nilai bilangan menjadi suatu string diantaranya ditunjukkan pada tabel di bawah

Fungsi	Prototipe	Keterangan
itoa	Stdlib.h	Mengkonversi int argumen menjadi nilai bertipe string
ltoa	Stdlib.h	Mengkonversi long int argumen menjadi nilai bertipe string
ultoa	Stdlib.h	Mengkonversi unsigned long argumen menjadi nilai bertipe string

Ketiga fungsi yang tercantum pada tabel di atas mempunyai tiga buah argumen.

- Argumen pertama berupa nilai yang akan dikonversi ke string
- Argumen kedua berupa variabel penerima string hasil konversi
- Argumen ketiga berupa basis bilangan yang digunakan.
Bisa diisi dengan nilai antara 2 sampai dengan 36.

Misalnya, apabila hasil didefinisikan sebagai berikut :

```
char hasil[24];
```

```
int
```

```
nilai= 2345;
```

Pernyataan :

```
itoa(nilai,hasil,10);
```

```
membuat hasil berisi "2345";
```

BAB III

POINTER

Pointer adalah variable yang berisi alamat memory sebagai nilainya dan berbeda dengan variable biasa yang berisi nilai tertentu. Dengan kata lain, pointer berisi alamat dari variable yang mempunyai nilai tertentu.

Dengan demikian, ada variabel yang secara langsung menunjuk ke suatu nilai tertentu, dan variabel yang secara tidak langsung menunjuk ke nilai.

Adapun bentuk umum dari pernyataan variabel pointer dalam C++ adalah :

Type *variabel-name

Dengan :

- Type adalah tipe dasar pointer
- Variabel name adalah nama variabel pointer
- * adalah variabel pada alamatnya yang ditentukan oleh operand.

Contoh :

```
Int *int_pointer;      // pointer to integer
Float *float_pointer;  // pointer to float
```

Contoh :

```
//Program : pointer.cpp
#include <stdio.h>
main()
{
    int a, *b;
    a=20;
    b=&a;

    printf(" Pointer b menunjukkan alamat =%p\n",b);
    printf(" Alamat tersebut berisi nilai :%d\n",*b);
}
```

```
//Program : pointer1.cpp
#include <iostream .h>
```

```
// cetak p dan *p void main(void)

{
    int v = 7, *p;
    p = &v;
    cout << " Nilai v = " << v << " dan *p = " << *p

    << "\nAlamatnya = " << p << "\n";
}
```

Bila program diatas dijalankan, maka hasilnya adalah sebagai berikut : Nilai v = 7 dan *p = 7 Alamatnya = efffb24

```
//Program:pointer2.cpp
#include <iostream.h>
```

```
int main ()
{
    int value1 = 5, value2 = 15;
    int * mypointer;
```

```
mypointer = &value1;
*mypointer = 10;
mypointer = &value2;
*mypointer = 20;
cout << "value1==" << value1 << "/ value2==" << value2;
return 0;
}
```

Bila program diatas dijalankan, maka hasilnya adalah sebagai berikut : "value1==" 10 << "/" value2==20

OPERATOR POINTER

Ada beberapa operator yang bisa digunakan dalam pointer. Operator tersebut adalah operator alamat (yang dilambangkan dengan simbol &) dan operator unary yang mengembalikan alamat dari operandnya.

Sebagai contoh, diasumsikan deklarasi sebagai

```
berikut : Int y = 5;
Int *yPtr;
```

Maka pernyataan : YPtr = &y;

Pernyataan ini mengandung arti bahwa alamat dari variabel y ditujukan kepada variabel pointer yPtr.

Contoh lain :

```
Int
balance,
value; Int
*balptr;
Balance = 3200; // step 1
Balptr=&balance; // step 2
Value=*balptr; // step 3
```

Contoh
diagram
:

	Step 1			Step 2			Step 3	
12		Balptr	12	100	Balptr	12	100	Balptr
100	3200	Balance	100	3200	Balance	100	3200	Balance
130		value	130		value	130	3200	value

Contoh :

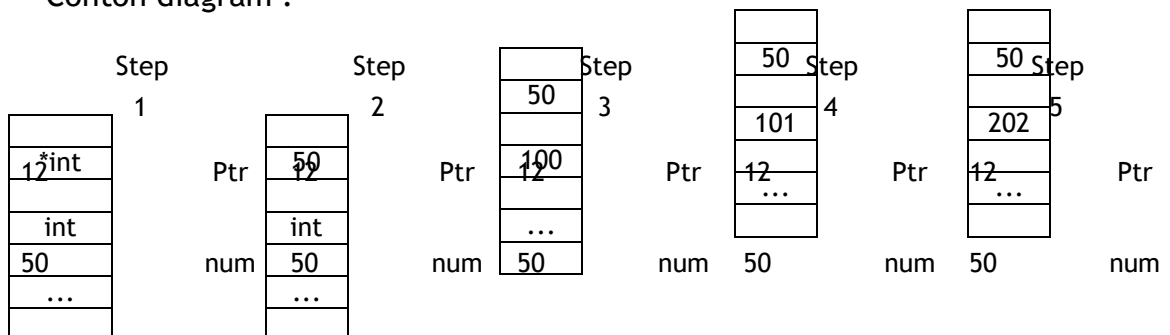
```
//Program:pointer3.cpp
#include <iostream.h>

int main()
{
    int *ptr, num;
    // Step 1 ptr = &num;
    // Step 2
    *ptr = 100;
    // Step 3 cout <<
    num << " ";
    (*ptr)++;
    // Step 4 cout <<
    num << " ";
    (*ptr)*=2;
    // Step 5 cout <<
    num << "\n";
    return 0;
}
```

Bila program dijalankan :

100
101
202

Contoh diagram :



EKSPRESI POINTER Pointer Aritmatika

Hanya 4 operator aritmatik dapat digunakan pada pointer ++, =, +, dan -.

Asumsi integer 32 bit.

Contoh :

//Program:pointer4.cpp

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

```
int main()
{
    int i[10],
    *i_ptr;
    double
    f[10],
    *f_ptr; int
    x;
    i_ptr = i; // i_ptr points to first element of i

    f_ptr = f; // f_ptr points to first
    element of f for(x=0; x<10; x++)
    cout << i_ptr+x << " " << f_ptr+x << "\n";

    return 0;
}
```

Bila program dijalankan :

0xeffffd9c	0xeffffd48
0xeffffda0	0xeffffd50
0xeffffda4	0xeffffd58
0xeffffda8	0xeffffd60
...	...

Pointer Perbandingan

Pointer dapat dibandingkan dengan menggunakan operator hubungan, seperti

!=, ==, <, dan >.

Contoh :

//Program:pointer5.cpp

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

```
int main()
```

```
{
```

```
    int num[10];
```

```
    int *start, *end;
```

```
    start = num;
```

```
    end = &num[9];
```

```
    while(start != end) {
```

```
        cout << "Masukkan bilangan sebanyak 9 data : ";
```

```
        cin >> *start;
```

```
        start++;
```

```
    }
```

```
    return 0;
```

```
}
```

Data yang akan dimasukkan sebanyak 9 buah data dan program tidak akan berhenti apabila belum sampai 9 buah data

POINTER VERSUS ARRAY

Array dan pointer adalah dua struktur data yang saling berkaitan satu dengan yang lain dalam C, dan dapat saling dipertukarkan penggunaannya. Karena array dapat didefinisikan sebagai pointer.

Contoh :

```
int *bPtr, b[5];
```

Dapat dibuat menjadi :

```
bPtr = b;  
bPtr = &b[0];
```

Berarti bPtr ditugaskan untuk menunjukkan ke alamat elemen pertama dari array b atau b[0].

Elemen array b[3]

dapat ditulis :

```
*(bPtr + 3)
```

Alamat &b[3]

dapat ditulis :

```
bPtr + 3
```

Deklarasi suatu variabel array x[] yang berisi nilai int

dapat ditulis :

```
int *x;
```

Variabel array ganda

dapat ditulis :

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

```
int *y [ ];
```

```
int *( *y);
```

Penggunaan pointer dan array untuk deklarasi variabel array ganda untuk menyimpan empat buah elemen yang masing-masing bertipe string (array dari karakter).

```
char *suit [4] = { "Hearts", "Diamonds", "Clubs", "Spades" };
```

Contoh :

```
//Program:pointer6.cpp  
#include <iostream.h>  
#include <stdio.h>
```

```
int main()
{
    char str[80];
    char token[80];
    char *str_ptr, *tk_ptr;
    cout << "Masukkan sebuah kalimat : ";
    gets(str);
    str_ptr = str;

    while(*str_ptr) {
        tk_ptr = token;

        while( *str_ptr != ' ' && *str_ptr ) {
            *tk_ptr
            =
            *str_ptr;
            tk_ptr++;
            ;
            str_ptr+
            +;
        }
        if(*str_ptr) str_ptr++;
        *tk_ptr = '\0';
        cout << token << endl;
    }
    return 0; }

```

//Program:pointer7.cpp
#include <iostream.h>
#include <stdio.h>

```
int main()
{
    char
    str[80
    ];
    char
    token
    [80];
    int i,
    j;
    cout << "Masukkan sebuah kalimat: ";

    gets(str);

    for(i=0; ; i++) {

```

```

        for(j=0; str[i] != ' ' && str[i]; j++, i++)
            token[j] = str[i];

    }return 0;

```

```

        token[j] = '\0';
    cout << token << '\n';
    if(!str[i]) break

```

Pada contoh program pointer 6 adalah program dengan pointer, sedangkan contoh program pointer 7 dengan menggunakan array. Hasil dari kedua program tersebut sama.

POINTER INDEX

Pointer tidak hanya dapat digunakan untuk mengakses elemen array, tetapi pointer juga dapat diindex seperti pada array.

Contoh :

```

//Program:pointer8.cpp
#include <iostream.h>
#include <ctype.h>

int main()
{
    char str[20] = "hello tom";
    char *p;
    int i;
    p = str;
    for(i=0; p[i]; i++)
        p[i] = toupper(p[i]);
    cout << p;
    return 0;
}

```

Bila program diatas dijalankan, maka hasilnya adalah sebagai berikut : HELLO TOM

Proses pengindexan pointer dapat dilihat pada variabel p yang menunjuk pada variabel str yang berisi data nama dengan panjang 20. Fungsi toupper memperjelas proses pengindexan

KONSTANTA STRING DAN POINTER

Konstanta string terlihat dalam program teks dan disimpan dalam tabel string serta setiap entry dalam tabel string, pointer string dibangkitkan.

Contoh :

```
//Program:pointer9.cpp
#include <iostream.h>
int main()
{
    char *s;
    s = "Pointers are fun to use.\n";
    cout << s;
    return 0;
}
```

Bila program diatas dijalankan, maka hasilnya adalah sebagai berikut : Pointers are fun to use.

ARRAY POINTER

Pointer dapat diarraykan seperti tipe data yang lain dalam C++. Untuk menyatakan sebuah array pi dari pointer sebanyak 10 buah data yang bertipe 10 integer, dapat ditulis :

```
int *pi [10];
```

Untuk menentukan alamat dari variabel integer disebut var ke elemen ketiga dari pointer array, dapat ditulis :

```
int var;  
pi [2] = &var
```

Contoh :

//Program:point10.cpp

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

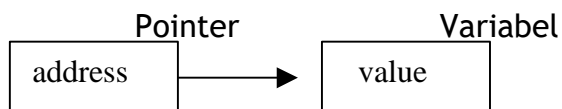
```
int main ()  
{  
    int numbers[5];  
    int *p;  
    p = numbers;  
    *p = 10;  
    p++; *p = 20;  
    p = &numbers[2];  
    *p = 30;  
    p = numbers + 3;  
    *p = 40;  
    p = numbers;  
    *(p+4) = 50;  
    for (int n=0; n<5; n++)  
        cout << numbers[n] << " , ";  
    return 0;  
}
```

Bila program diatas dijalankan, maka hasilnya adalah sebagai berikut :

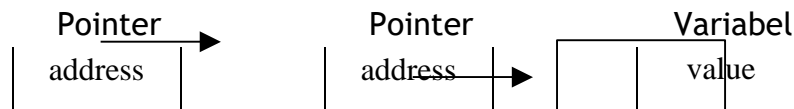
10, 20, 30, 40, 50,

POINTER DALAM POINTER

C++ memperbolehkan penggunaan pointer dalam pointer yang berisi data yang sama atau berbeda. Dalam kondisi pointer biasa atau pointer tunggal, diagramnya adalah :



Untuk pointer dalam pointer, diagramnya adalah :

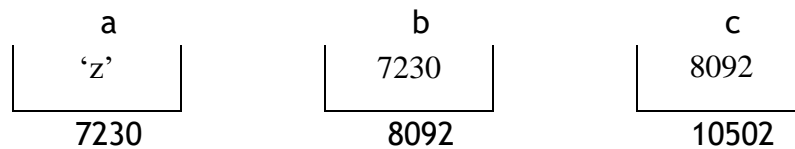


Contoh :

```

char a;
char *b;
char **c;
a = 'z'; b = &a; c = &b
  
```

Dengan misal data acak pada memori 7230, 8092, dan 10502, maka diagramnya adalah :



Dari diagram dapat disimpulkan :

- c adalah sebuah variabel dengan tipe (char**) yang berisi 8092
- *c adalah sebuah variabel dengan tipe (char*) yang berisi 7230
- **c adalah sebuah variabel dengan tipe (char) yang berisi 'z'

Contoh :

//Program:point12.cpp

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

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int main()
{
    int x, *p, **q;
    x = 10; p = &x; q = &p;
    cout << **q; // prints the
    value of x return 0;
}
```

Bila program diatas dijalankan, maka hasilnya adalah sebagai berikut :
10

SOAL LATIHAN : Soal 1

Program untuk menghitung banyaknya karakter yang dimasukkan dengan menggunakan pointer.

Save dengan nama file : po1_nim (4 digit nim terakhir)

Soal 2

Program untuk merubah karakter yang dimasukkan dari huruf kecil menjadi huruf besar.

Save dengan nama file : po2_nim (4 digit nim terakhir)

BAB IV STRUKTUR

Struktur adalah sekumpulan variabel yang masing-masing dapat berbeda tipe, dan dikelompokkan ke dalam satu nama (menurut Pascal, struktur juga dikenal sebagai record). Struktur membantu mengatur data-data yang rumit, khususnya dalam program yang besar, karena struktur membiarkan sekelompok variabel diperlakukan sebagai satu unit daripada sebagai entity yang terpisah.

Salah satu contoh struktur tradisional adalah record daftar gaji karyawan, dimana karyawan digambarkan dengan susunan lambang seperti nama, alamat, nomor jaminan sosial, gaji dan sebagainya. Beberapa dari lambang tersebut biasanya berupa struktur, nama mempunyai komponen begitu juga alamat dan gaji.

Struktur ini sering digunakan untuk mendefinisikan suatu record data yang disimpan di dalam file. Struktur termasuk ke dalam tipe data yang dibangkitkan (derived data type), yang disusun dengan menggunakan obyek tipe lain.

Contoh :

```
struct mhs
{
    char *nama;
    char *nim;
    int tts, tas;
    float akhir;
    char aksara;
}
```

Kata kunci struct menunjukkan definisi struktur, dan identitas mhs menunjukkan structure tag. Dengan demikian terdapat tipe data baru bernama struct mhs, yang terdiri dari nama mahasiswa, nilai tes tengah semester, tes akhir semester, nilai akhir, dan huruf aksara, yang masing-masing disebut dengan field.

Dapat dituliskan
dengan :

```
struct mhs x, y[100], *z;
```

Variabel x adalah variabel tunggal, y adalah variabel array dengan 100 lokasi memori, dan z adalah variabel pointer, yang kesemuanya masing-masing berisi field di atas. Jadi, variabel y adalah daftar nama, nilai tts, tas, akhir, dan huruf aksara dari 100 mahasiswa.

Sehingga
dapat ditulis :

```
struct mhs
{
    char *nama;
    char *nim;
    int tts, tas;
    float akhir;
    char aksara;
} x, y[100], *z;
```

Inisialisasi juga dapat dilakukan
dengan :

```
struct mhs x = { "Garfield", 80, 60, 76.8, 'A' };
```

Untuk mengakses anggota dari struktur digunakan salah satu dari dua operator, yaitu operator titik (.), atau operator panah (->) tergantung tipe variabel yang dideklarasikan. Jika variabel tunggal (misal x) maka

digunakan operator titik, sedangkan jika variabel pointer (misal z) digunakan operator panah.

Contoh :

```
struct point
{
    int x;
    int y;
};
```

kata kunci struct mengenalkan deklarasi struktur yang mana deklarasi list terlampir di kurung kurawal { }. Nama pilihan yang disebut structure tag mengikuti kata struct.

Deklarasi struktur yang tidak diikuti oleh variabel list tidak menyediakan tempat penyimpanan; deklarasi struktur hanya menjelaskan template atau bentuk struktur. Kalau deklarasi di tag, tag dapat digunakan dalam definisi contoh struktur. Sebagai contoh, memberikan deklarasi point diatas.

```
struct point pt;
```

Variabel pt yang berupa struktur tipe struct poin. Sebuah struktur dapat diletakan di depan dengan mengikuti definisinya dengan daftar inisialisasi, masing-masing adalah lambang konstanta

STRUKTUR DAN FUNGSI

Operasi yang sering diterapkan pada struktur adalah proses menyalin atau menunjukkan struktur sebagai unit, menggunakan alamatnya dan mengakses anggotanya. Copy dan assignment mencakup memberi argumen ke fungsi dan menghasilkan nilai dari fungsinya juga. Struktur tidak bisa dibandingkan.

Struktur dapat diletakkan di awal oleh daftar value konstanta dan otomatis juga dapat ditempatkan di awal oleh operasi assignment. Sebuah struktur otomatis mungkin juga diletakkan di depan oleh tugas atau oleh panggilan fungsi yang menghasilkan struktur jenis yang tepat.

Untuk menghubungkan nama struktur dan nama anggota digunakan simbol “.”

Contoh :

```
/*Program : struct1.cpp*/
#include <stdio.h>
struct time
{
    int jam;
    int min;
};
struct rencana {
    struct time awal;
    struct time akhir;
    int y;
    int z;
};
struct rencana kerja = { 11,22,33,44,5,6 };

funct(struct rencana oo);
```

```

func(struct rencana oo);
main()
{
    kerja.akhir.min = 40;
    kerja.z = 66;
    printf("proses main sebelum ke fungsi\n%d %d %d %d %d %d\n",
        kerja.awal.jam, kerja.awal.min, kerja.akhir.jam, kerja.akhir.min, kerja.y, kerja.z);

    func(kerja); /* pengiriman struktur kerja ke fungsi */

    printf("proses main sesudah ke fungsi\n%d %d %d %d %d %d\n", kerja.awal.jam,
        kerja.awal.min, kerja.akhir.jam, kerja.akhir.min, kerja.y, kerja.z);
}

```

```

func(struct rencana oo)
/* nilai struktur kerja disalinkan ke oo */
{
    printf("dalam fungsi (a)\n%d %d %d %d %d %d\n", oo.awal.jam,
        oo.awal.min, oo.akhir.jam, oo.akhir.min, oo.y, oo.z);

    oo.awal.jam = 111;
    oo.y = 555; /* ubah nilai dalam fungsi */

    printf("dalam fungsi (a)\n%d %d %d %d %d %d\n", oo.awal.jam,
        oo.awal.min, oo.akhir.jam, oo.akhir.min, oo.y, oo.z);
}

```

Bila program dijalankan maka :

```

proses main sebelum ke fungsi
11 22 33 40 5 66 dalam fungsi (a)
11 22 33 40 5 66 dalam fungsi (a)
111 22 33 40 555 66
proses main sesudah ke fungsi
11 22 33 40 5 66

```

ARRAY DALAM STRUKTUR

Array disini berfungsi untuk menyimpan nama dan bilangan bulat yang akan digunakan dalam proses perhitungan.

Contoh :

// Program : struct2.cpp

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

```
struct movies_t {
    char title [50];
    int year;
} mine, yours;
```

```
void printmovie (movies_t movie);
```

```
int main ()
{
    char buffer [50];

    strcpy (mine.title, "Finding Nemo");
    mine.year = 2003;

    cout << "Masukkan judul film favorit: ";
    cin.getline
    (yours.title,50);
    cout <<
    "Masukkan tahun:
    "; cin.getline
    (buffer,50);
    yours.year = atoi
    (buffer);

    cout << "Judul film favorit yang ada:\n ";
```

```
    printmovie (mine);  
    cout << "Judul film favorit kamu adalah:\n ";  
    printmovie (yours);  
    return 0;  
}
```

```
void printmovie (movies_t movie)  
{  
    cout << movie.title;  
    cout << " (" << movie.year << ")\n";  
}
```

Bila program diatas dijalankan maka hasilnya

adalah : Masukkan judul file favorit :

Avp

Masukkan tahun : 2004

Judul film favorit

yang ada : Finding

Nemo (2003)

Judul film favorit kamu

adalah : Avp (2004)

Dapat dilihat deklarasi array dalam struktur terletak pada char title[50]; dimana dalam title dapat menyimpan judul film sebanyak 50 tempat dan title ini terletak pada

```
struct movies_t {  
    char title [50];  
    int year;  
} mine, yours;
```


Contoh :

```
// Program : struct3.cpp
// array of structures
#include <iostream.h>
#include <stdlib.h>
#define N_MOVIES 5 struct movies_t {
    char title [50];
    int year;
} films [N_MOVIES];

void printmovie (movies_t movie);

int main ()
{
    char buffer [50];
    int n;
    for (n=0; n<N_MOVIES; n++)
    {

        cout << "Masukkan judul film: ";
        cin.getline
        (films[n].title,50);
        cout << "Masukkan
        tahun : "; cin.getline
        (buffer,50);
        films[n].year = atoi
        (buffer);
    }
    cout << "\nFilm yang menjadi favorit kamu:\n";
    for (n=0; n<N_MOVIES; n++)
        printmovie (films[n]);
    return 0;
}

void printmovie (movies_t movie)
{
    cout << movie.title;
    cout << " (" << movie.year << ")\n";
}
```

Bila program diatas dijalankan maka hasilnya

adalah : Masukkan judul file favorit :

Matrix

Masukkan tahun : 2002

Masukkan judul file favorit : Alien Vs Predator

Masukkan tahun : 2004

...

Film yang menjadi

favorit kamu : Matrix

(2002)

Alien Vs Predator (2004)

Deklarasi array dalam struktur terletak pada char title [50]; dan
#define

N_Movies 5, sehingga 5 buah data film harus
dimasukkan.

POINTER DALAM STRUKTUR

Misalkan sebuah pointer yaitu `ptpelajar`, yang menunjuk kepada sebuah data yang mempunyai struktur `PELAJAR` seperti berikut :

```
struct PELAJAR *ptpelajar;
```

Seperti pada pointer yang lain, deklarasi tersebut tidak menyediakan sebarang tempat untuk record `PELAJAR`. Perlu dibuat record baru yang fungsinya menggantikan pointer. Misalnya `pelajar_baru`.

```
ptpelajar = &pelajar_baru;
```

Dengan kondisi tersebut, pointer `ptpelajar` boleh digunakan untuk menggantikan tempat alamat `pelajar_baru`, dan pointer `ptpelajar` ini ditunjukkan dengan menggunakan simbol `->`.

Contoh :

```
ptpelajar->nama = Jill Valentine;  
ptpelajar->nim = 672009001;  
ptpelajar->fakultas = Teknologi Informasi;  
ptpelajar->tahun = 2009;  
ptpelajar->alamat = Salatiga;
```

Sama dengan :

```
*ptpelajar.nama = Jill Valentine;  
*ptpelajar.nim = 672009001;  
*ptpelajar.fakultas = Teknologi Informasi;  
*ptpelajar.tahun = 2009;  
*ptpelajar.alamat = Salatiga;
```

Contoh :

// Program : struct4.cpp

```
#include <iostream.h>
struct time {
    int jam;
    int min;
};
struct rencana {
    struct time *awal; /* penunjuk bagi struktur */
    struct time *akhir;
};
struct time jk = { 1,2 };
struct time kl = { 3,4 };
struct rencana kerja = { &jk,&kl };

main()
{
    kerja.akhir.min = 40;
    kerja.z = 66;
    printf("proses main sebelum ke fungsi\n%d %d %d %d %d %d\n",
        kerja.awal.jam, kerja.awal.min, kerja.akhir.jam, kerja.akhir.min, kerja.v.kerja.z);

    funct(kerja); /* pengiriman struktur kerja ke fungsi */

    printf("proses main sesudah ke fungsi\n%d %d %d %d %d %d\n", kerja.awal.jam,
        kerja.awal.min, kerja.akhir.jam, kerja.akhir.min, kerja.v.kerja.z);
}
```

Bila program diatas dijalankan maka hasilnya adalah :

1 2 3 37

STRUKTUR YANG MENUNJUK DIRINYA SENDIRI

Struktur yang mempunyai sifat menunjuk dirinya sendiri dapat dilihat pada contoh deklarasi berikut ini :

```
struct tnode {
    char *perkataan;
    int *perkiraan;
    struct tnode *kiri;
    struct tnode *kanan;
};
```

TYPEDEF

Kata kunci typedef merupakan mekanisme untuk membuat sinonim atau alias dari tipe data yang telah didefinisikan sebelumnya.

Contoh :

```
typedef struct mhs MHS;
```

Dari deklarasi tersebut dapat didefinisikan sebuah tipe data baru bernama MHS sebagai sinonim untuk struct mhs. Pernyataan struct mhs dapat diganti dengan MHS saja.

Contoh :

```
typedef int PANJANG;
```

Jenis PANJANG boleh digunakan untuk deklarasi variabel yang lain.

Contoh :

```
PANJANG len, maxlen;  
PANJANG *lengths[];
```

Sama dengan :

```
int len, maxlen;  
int *lengths[];
```

Contoh :

```
// Program : struct5.cpp  
#include <stdio.h>  
main()  
{  
    static struct s1 {  
        char c[4], *s;
```

```
} s1 = { "abcd", "fghi" };

static struct s2 { char *cp; struct S1 ss1;
} s2 = { "klmn", { "pqrs", "uvwxy" } };

printf("s1.c[2] = %c, *s1.s = %c\n", s1.c[2], *s1.s);
printf("s1.c = %s s1.s = %s\n", s1.c, s1.s);
printf(" s2.cp = %s s2.ss1.s = %s\n", s2.cp, s2.ss1.s);
printf("++s2.cp = %s ++s2.ss1.s = %s\n", ++s2.cp, ++s2.ss1.s);
}
```

Bila program tersebut dijalankan maka :

```
s1.c[2] = c, *s1.s = f s1.c = abcd s1.s = fghi
s2.cp = klmn s2.ss1.s = uvwxy
++s2.cp = lmn ++s2.ss1.s = vwxy
```

Penggunaan typedef terletak pada variabel s1 dan s2 yang mempunyai panjang karakter yang berbeda dan nantinya pada waktu dicetak akan menghasilkan output sesuai kondisi pada program.

UNION

Sama seperti struct, union juga merupakan tipe data yang dibangkitkan, dimana anggotanya menggunakan secara bersama ruang penyimpanan memori yang sama, berbeda dengan struktur yang masing-masing variabel menempati lokasi memori yang berbeda.

Jumlah bytes yang digunakan untuk menyimpan union adalah sedikitnya cukup untuk menyimpan data terbesar yang ditangani. Tipe union umumnya digunakan untuk menangani satu, dua, atau tiga variabel dengan tipe yang mirip.

Contoh :

```
//Program:struct6.cpp
#include <iostream.h>
typedef union int_or_float {
    int n;
    float x;
} number;

Main()

{

    number temp;
    temp = 4444;
    Cout << "(a) temp.n = " << temp.n << " temp.x = " << temp.x << endl;
    temp.x = 4444.0;

    cout << "(b) temp.n = " << temp.n << " temp.x = " << temp.x << endl;
    temp.n = 4444;
    cout << "(c) temp.n = " << temp.n << " temp.x = " << temp.x << endl

}
```

Bila program dijalankan, maka :

```
(a) temp.n = 4444   temp.x = 0.574484 (b) temp.n = -8192
temp.x = 4444
(c) temp.n = 4444   temp.x = 4418.169922
```

Output pada baris (b), nilai temp.n berubah karena tempatnya telah digunakan untuk menempatkan temp.x. Hal sebaliknya terjadi kepada nilai temp.x pada baris .

Output program akan berbeda jika deklarasi union tadi diganti dengan deklarasi struktur berikut :

```
typedef struct int_or_float {  
    int n;  
    float x;  
} number;
```

Hasilnya :

(a) temp.n = 4444 temp.x = -0.00000
(b) temp.n = 4444 temp.x = 4444.00000 (c) temp.n = 4444
temp.x = 4444.00000

ENUMERASI

C++ menyediakan tipe data yang dapat didefinisikan oleh pemrogram disebut dengan enumerasi. Enumerasi, didefinisikan dengan menggunakan kata kunci enum, adalah sekumpulan konstanta integer yang direpresentasikan dengan identifikasi tertentu.

Nilai dalam enum dimulai dari 0, dapat diubah dengan nilai lainnya, dan menaik dengan penambahan 1 untuk nilai selanjutnya.

Contoh :

```
enum bulan {JAN, PEB, MAR, APR, MEI, JUN, JUL, AGU, SEP,
            OKT, NOP, DES};
```

Deklarasi tersebut akan menciptakan tipe baru yaitu enum bulan, yang secara otomatis menunjukkan deret nilai 0 untuk JAN hingga 11 untuk DES.

Nilai bulan ini dapat diubah menjadi 1 hingga 12 dengan cara :

```
enum bulan {JAN = 1, PEB, MAR, APR, MEI, JUN, JUL, AGU,
            SEP, OKT, NOP, DES};
```

Contoh :

```
/* Program : struct7.cpp */
#include <stdio.h>
```

```
enum bulan {JAN = 1, PEB, MAR, APR, MEI, JUN, JUL, AGU, SEP, OKT,
            NOP, DES};
```

```
main() {
    enum bulan Bulan;
    char *namaBulan[] = { "", "Januari", "Pebruari", "Maret", "April",
                          "Mei", "Juni", "Juli", "Agustus", "September", "Oktober",
                          "Nopember", "Desember" };
}
```

```
for ( Bulan = JAN ; Bulan <= 12 ; Bulan++ )
    printf( "%2d%11s\n", Bulan, namaBulan[Bulan] );
return 0;
}
```

Bila program tersebut dijalankan maka :

1	Januari
2	Februari
3	Maret
4	April
5	Mei
6	Juni
7	Juli
8	Agustus
9	September
10	Oktober
11	November
12	Desember

CONTOH SOAL : Soal 1

Buatlah program dengan struktur untuk menampilkan data karyawan yang berjumlah 5 orang lengkap dengan nama dan no Id-nya.

Contoh ada di file : dataid.exe

Save dengan nama file : st1_nim (4 digit nim terakhir)

BAB V SORT

Sort adalah suatu proses pengurutan data yang sebelumnya disusun secara acak atau tidak teratur menjadi urut dan teratur menurut suatu aturan tertentu.

Biasanya pengurutan terbagi menjadi
2 yaitu :

- ¾ Ascending (pengurutan dari karakter / angka kecil ke karakter / angka besar)
- ¾ Descending (pengurutan dari karakter / angka besar ke karakter / angka kecil)

Ada banyak cara yang dapat dilakukan untuk melakukan proses pengurutan dari paling tinggi ke paling rendah atau sebaliknya. Untuk masalah pengurutan pada array kita tidak dapat langsung menukar isi dari variabel yang ada, tetapi menggunakan metode penukaran (swap).

Contoh :

Data yang terdiri dari nilai dengan array, nilai[1] = 10 dan nilai[2] = 8 akan ditukar isi nilainya sehingga akan menghasilkan nilai[1] = 8 dan nilai[2] = 10.

Proses penukaran tidak dapat langsung dilakukan dengan cara :

```
nilai[1] = nilai[2];  
nilai[2] = nilai[1];
```

Cara yang tepat adalah :

```
temp = nilai[1];  
nilai[1] = nilai[2];  
nilai[2] = temp;
```

Untuk melakukan proses pengurutan dapat menggunakan beberapa metode antara lain :

1. Bubble Sort
2. Selection Sort
3. Quick Sort

METODE SORTING

1. Bubble Sort

Bubble sort adalah suatu metode pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya. Apabila elemen yang sekarang lebih besar dari elemen yang berikutnya, maka posisinya akan ditukar, bila tidak maka posisi akan tetap.

Misalkan data sebagai
berikut :

5, 34, 32, 25, 75, 42, 22, 2

Pengurutan akan dilakukan dengan mengurutkan 8 data dari yang terkecil sampai yang terbesar.

Dimulai dengan dua angka pertama yaitu angka 5 dan 34, untuk pengurutan dari kecil ke besar maka akan diperoleh posisi tidak berubah karena $5 < 34$.

- Langkah berikutnya akan membandingkan angka 34 dan 32.

Karena angka

$34 > 32$ maka akan terjadi pertukaran posisi, sehingga hasil urutan datanya menjadi 5, 32, 34, 25, 75, 42, 22, 2.

- $34 > 25 = 25, 34$ (berubah)
- $34 < 75 = 34, 75$ (tetap)
- $75 > 42 = 42, 75$ (berubah)
- $75 > 22 = 22, 75$ (berubah)
- $75 > 2 = 2, 75$ (berubah)

Hasil sementara menjadi 5, 32, 25, 34, 42, 22, 2, 75

Langkah kedua :

- $5 < 32 = 5, 32$ (tetap)
- $32 > 25 = 25, 32$ (berubah)
- $32 < 34 = 32, 34$ (tetap)
- $34 < 42 = 34, 42$ (tetap)
- $42 > 22 = 22, 42$ (berubah)
- $42 > 2 = 2, 42$ (berubah)
- $42 < 75 = 42, 75$ (tetap)

Hasil sementara menjadi 5, 25, 32, 34, 22, 2, 42, 75

Langkah ketiga :

- $5 < 25 = 5, 25$ (tetap)

- $25 < 32 = 25, 32$ (tetap)
- $32 < 34 = 32, 34$ (tetap)
- $34 > 22 = 22, 34$ (berubah)
- $34 > 2 = 2, 34$ (berubah)
- $34 < 42 = 34, 42$ (tetap)
- $42 < 75 = 42, 75$ (tetap)

Hasil sementara menjadi 5, 25, 32, 22, 2, 34, 42, 75

Sampai langkah terakhir yaitu langkah ketujuh :

- $5 > 2 = 2, 5$ (berubah)
- $5 < 22 = 5, 22$ (tetap)
- $22 < 25 = 22, 25$ (tetap)
- $25 < 32 = 25, 32$ (tetap)
- $32 < 34 = 32, 34$ (tetap)
- $34 < 42 = 34, 42$ (tetap)
- $42 < 75 = 42, 75$ (tetap)

Hasil sementara menjadi 2, 5, 22, 25, 32, 34, 42, 75

Proses pengurutan data tersebut membutuhkan tujuh langkah atau tujuh kali perulangan.

Contoh :

//Program:buble.cpp

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

```
#include <iomanip.h>
```

```
void main()
```

```
{
```

```
    int NumList[8] = {5, 34, 32, 25, 75, 42, 22, 2};
```

```
    int Swap;
```

```
    cout<<"Data sebelum diurutkan: \n";
```

```

for(int ctr=0; ctr<8; ctr++)
{
    cout<< setw( 3 ) << NumList[ctr];
}
cout<< "\n\n";
for(int i=0; i<7; i++)
    for(int ii=0; ii<7; ii++)
        if (NumList[ii] > NumList[ii + 1])
        {
            Swap = NumList[ii];
            NumList[ii] =
            NumList[ii + 1];
            NumList[ii + 1] =
            Swap;
        }
cout<< "Data setelah diurutkan: \n";

for (int iii=0; iii<8; iii++)
    cout<< setw( 3 ) << NumList[iii];
cout<< endl << endl;
}
    
```

Bila program tersebut dijalankan maka : Data sebelum diurutkan :

5, 34, 32, 25, 75,
42, 22, 2

Data setelah diurutkan :

2, 5, 22, 25, 32,
34, 42, 75

Data pada program diatas akan diurutkan menurut ascending atau dari kecil ke besar.

2. Selection Sort

Selection sort adalah suatu metode pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya sampai ke elemen yang terakhir. Jika ditemukan elemen lain yang

lebih kecil dari elemen sekarang maka dicatat posisinya dan langsung ditukar.

Misalkan data sebagai berikut :

5, 34, 32, 25, 75, 42, 22, 2

Data tersebut akan diurutkan dengan ascending atau dari kecil ke besar.

Langkah pertama :

Posisi	:	1	2	3	4	5	6	7	8
Data	:	5	34	32	25	75	42	22	2

Pembanding Posisi

- $5 < 34$ 1
- $5 < 32$ 1
- $5 < 25$ 1
- $5 < 75$ 1
- $5 < 42$ 1
- $5 < 22$ 1
- $5 > 2$ 8

Tukar data pada posisi 1 dengan data posisi 8.

Hasil sementara menjadi 2, 34, 32, 25, 75, 42, 22, 5

Langkah kedua :

Posisi	:	1	2	3	4	5	6	7	8
Data	:	2	34	32	25	75	42	22	5

Pembanding Posisi

- $34 > 32$ 3

- $32 > 25$ 4
- $25 < 75$ 4
- $25 < 42$ 4
- $25 > 22$ 7
- $22 > 2$ 8

Tukar data pada posisi 2 dengan data posisi 8.

Hasil sementara menjadi 2, 5, 32, 25, 75, 42, 22, 34

Langkah ketiga :

Posisi	:	1	2	3	4	5	6	7	8
Data	:	2	5	32	25	75	42	22	34

Pembandingan		Posisi
--------------	--	--------

- $32 > 25$ 4
- $25 > 75$ 4
- $25 < 42$ 4
- $25 > 22$ 7
- $22 < 34$ 7

Tukar data pada posisi 3 dengan data posisi 7.

Hasil sementara menjadi 2, 5, 22, 25, 75, 42, 32, 34

Langkah keenam (langkah terakhir) :

Posisi	:	1	2	3	4	5	6	7	8
Data	:	2	5	22	25	32	34	75	42

Pembandingan		Posisi
--------------	--	--------

- $75 > 42$ 8

Tukar data pada posisi 7 dengan data posisi 8.

Hasil sementara menjadi 2, 5, 22, 25, 32, 34, 42, 75

Proses pengurutan data tersebut membutuhkan enam langkah atau enam kali perulangan.

Contoh :

```
void SelectionSort (int Array[ ], const int Size)
{
    int i, j, smallest, temp;
    for (i=0; i<Size; i++)
    {
        smallest=i;
        for (j=i; j<Size; j++)
        {
            if (array[smallest]>array[j])
            {
                smallest=j;
            }
        }

        temp=array[i];
        array[i]=array[smallest];
        array[smallest]=temp;
    }
}
```

3. Quick Sort

Quick sort adalah suatu metode pengurutan yang membandingkan suatu elemen (pivot) dengan elemen yang lain dan menyusunnya sedemikian rupa sehingga elemen yang lain yang lebih kecil daripada pivot terletak disebelah kiri pivot sedangkan elemen yang lebih besar dari pivot diletakkan di sebelah kanan pivot.

Sehingga akan terbentuk dua sub list yaitu yang terletak disebelah kiri pivot dan sebelah kanan pivot.

List yang sebelah kiri pivot juga diterapkan aturan seperti pivot, yaitu membandingkan dengan elemen yang lain. Jika lebih kecil akan diletakkan di sebelah kiri, jika lebih besar akan diletakkan di sebelah kanan.

Contoh :

```
void QuickSort (array A, int L, int N)
{
```

```
    if L < N
    {
        M := Partition (A, L, N)
        QuickSort (A, L, M-1)
        QuickSort (A, M+1, N)
    }
endif
```

```
void Partition (array A, int L, int N)
{
    select M, where L <= M <= N
    reorder A(L) ... A(N) so that L < M implies A(L) <= A(M),
    and L > M
    implies A(L) >= A(M)
    return M
}
```

Contoh :

Data yang akan diurutkan adalah : 20, 10, 15, 5, 8, 3

Bilangan yang terletak diantara kurung buka dan kurung tutup
adalah pivot i bergerak dari kiri ke kanan sampai mendapat
nilai \geq pivot
j bergerak dari kanan ke kiri sampai mendapat nilai $<$ pivot

Langkah 1

posisi	1	2	3	4	5	6
data	20	10	(15)	5	8	3
	i					j

i berhenti pada posisi 1 karena langsung mendapatkan nilai yang lebih besar dari pivot (15) yaitu 20

j berhenti pada posisi 6 karena langsung mendapatkan nilai yang lebih kecil dari pivot (15) yaitu 3

karena $i < j$ maka data yang ditunjuk oleh i ditukar dengan data yang ditunjuk j, sehingga data berubah menjadi :

3 10 15 5 8 20

Langkah 2

posisi	1	2	3	4	5	6
data	3	10	(15)	5	8	20
			i		j	

i berhenti pada posisi 3 karena tidak dapat menemukan nilai yang lebih besar dari pivot (15)

j berhenti pada posisi 5 karena langsung mendapatkan nilai yang lebih

kecil dari pivot (15) yaitu 8

karena $i < j$ maka data yang ditunjuk oleh i ditukar dengan data yang ditunjuk j, sehingga data berubah menjadi :

3 10 8 5 15 20

Langkah 3

posisi	1	2	3	4	5	6
data	3	10	(8)	5	15	20
					i	j

i berhenti pada posisi 2 karena langsung mendapatkan nilai yang lebih besar dari pivot (8) yaitu 10

j berhenti pada posisi 4 karena langsung mendapatkan nilai yang lebih kecil dari pivot (8) yaitu 5

karena $i < j$ maka data yang ditunjuk oleh i ditukar dengan data yang ditunjuk j, sehingga data berubah menjadi :

3 5 8 10 15 20

Hasil akhirnya adalah : 3 5 8 10 15 20

CONTOH SOAL : Soal 1

Buatlah program untuk melakukan pengurutan data secara menurun (dari besar ke kecil) dengan menukar data berikut menggunakan bubble sort.

Data : 21, 83, 42, 11, 10, 9, 3, 20, 102, 27, 15, 92, 2

Bila program dijalankan maka :

102, 92, 83, 42, 27, 21, 20, 15, 11, 10, 9, 3, 2

BAB VI LINKED LIST

Konsep dasar struktur data dinamis adalah alokasi memori yang dilakukan secara dinamis. Pada konsep ini, terdapat suatu struktur yang disebut dengan struktur referensi diri (self-referential structure), mempunyai anggota pointer yang menunjuk ke struktur yang sama dengan dirinya sendiri.

Struktur data dinamis sederhana dapat dibagi menjadi empat jenis, yaitu :

1. Linked list
2. Stack
3. Queue
4. Binary tree

Definisi ini dapat dituliskan secara sederhana dengan struktur :

```
struct node {  
    int info;  
    struct node *nextPtr;  
}
```

Struktur ini mempunyai dua anggota, yaitu bilangan bulat info dan pointer nextPtr. Variabel pointer nextPtr ini menunjuk ke struktur bertipe struct node, yang sama dengan deklarasi induknya.

Untuk membuat dan menangani struktur data dinamis dibutuhkan alokasi

memori dinamis agar tersedia ruang bagi node-node yang ada. Fungsi malloc, free, dan operator sizeof banyak digunakan dalam alokasi memori dinamis ini.

Contoh :

```
typedef struct node NODE;  
typedef NODE *NODEPTR;  
NODEPTR newPtr = malloc (sizeof (NODE));  
newPtr -> info = 15;  
newPtr -> nextPtr = NULL;
```

Fungsi free digunakan untuk menghapus memori yang telah digunakan untuk node yang ditunjuk oleh pointer tertentu. Jadi free (newPtr); akan menghapus memori tempat node yang ditunjuk oleh newPtr.

DEFINISI LINKED LIST

Linked list adalah suatu cara untuk menyimpan data dengan struktur sehingga dapat secara otomatis menciptakan suatu tempat baru untuk menyimpan data yang diperlukan. Program akan berisi suatu struct atau definisi kelas yang berisi variabel yang memegang informasi yang ada didalamnya, dan mempunyai suatu pointer yang menunjuk ke suatu struct sesuai dengan tipe datanya.

Struktur dinamis ini mempunyai beberapa keuntungan dibanding struktur array yang bersifat statis. Struktur ini lebih dinamis, karena banyaknya elemen dengan mudah ditambah atau dikurangi, berbeda dengan array yang ukurannya bersifat tetap.

Manipulasi setiap elemen seperti menyisipkan, menghapus, maupun menambah dapat dilakukan dengan lebih mudah.

Contoh :

//Program:link1.cpp

```
#include <stdio.h>
```

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

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

```
typedef struct nod {  
    int data;  
    struct nod *next;  
} NOD, *NODPTR;
```

```
void CiptaSenarai(NODPTR *s)  
{  
    *s = NULL;  
}
```

```
NODPTR NodBaru(int m)  
{  
    NODPTR n;  
  
    n = (NODPTR) malloc(sizeof(NOD));  
    if(n != NULL) {  
        n->data = m;  
        n->next = NULL;  
    }  
    return n;  
}
```

```
void SisipSenarai (NODPTR *s, NODPTR t, NODPTR p)  
{  
    if (p==NULL) {  
        t->next = *s;  
        *s = t;  
    }
```



```
    }
    else {
        t -> next = p -> next;
        p -> next
    }
}

void CetakSenarai (NODPTR s)
{
    NODPTR ps;
    for (ps = s; ps != NULL; ps = ps -> next)
        printf("%d -->", ps -> data);
    printf("NULL\n");
}

int main ()
{
    NODPTR pel; NODPTR n;

    CiptaSenarai(&pel); n = NodBaru(55);
    SisipSenarai(&pel, n, NULL);

    n= NodBaru(75); SisipSenarai(&pel, n, NULL);
    CetakSenarai(pel);
    return 0;
}
```

Bila program dijalankan maka :
75->55->NULL

TEKNIK DALAM LINKED LIST

Pengulangan Linked List

Secara umum pengulangan ini dikenal sebagai while loop. Kepala pointer (head pointer) dikopikan dalam variabel lokal current yang kemudian dilakukan perulangan dalam linked list. Hasil akhir dalam linked list dengan `current!=NULL`. Pointer lanjut dengan `current=current->next`.

Contoh :

```
// Return the number of nodes in a list (while-loop version)
int Length(struct node * head) {
    int count = 0;
    struct node* current = head;
    while (current !=
        NULL) {
        count++;
        current=current->next;
    }
    return (count);
}
```

Mengubah Pointer Dengan Referensi Pointer

Banyak fungsi pada linked list perlu untuk merubah pointer kepala. Pointer ke pointer disebut dengan “reference pointer”.

Langkah utamanya dalam teknik ini adalah :

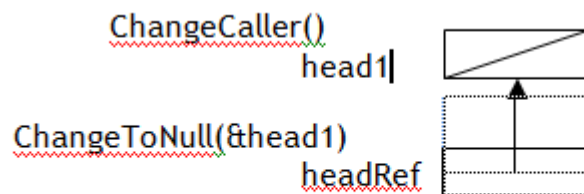
- Merancang fungsi untuk mengambil pointer ke pointer kepala. Untuk melewati pointer ke “value of interest” yang dibutuhkan untuk diubah. Untuk mengubah struct node*, lewati struct node**.

- Gunakan ‘&’ dalam panggilan untuk menghitung dan melewati pointer ke value of interest.
- Gunakan ‘*’ pada parameter dalam fungsi pemanggil untuk mengakses dan mengubah value of interest.

Contoh :

```
void ChangeToNull (struct node** headRef)
*headRef=NULL;
void ChangeCaller() { struct node* head1; struct node* head2;
    ChangeToNull (&head1); ChangeToNull (&head2);
}
```

Fungsi sederhana tersebut akan membuat pointer kepala ke NULL dengan menggunakan parameter reference. Gambar berikut menunjukkan bagaimana pointer headRef dalam ChangeToNull menunjuk ke head1 pada Change Caller.



Pointer headRef dalam ChangeToNull menunjuk ke head1

Membuat Kepala Senarai Dengan Perintah Push()

Cara termudah untuk membuat sebuah senarai dengan menambah node pada “akhir kepala” adalah dengan push().

Contoh :

```
struct node*
AddAtHead() {
    struct node* head = NULL;
    int i;
    for ( i=1; i<6; i++) {
```

```

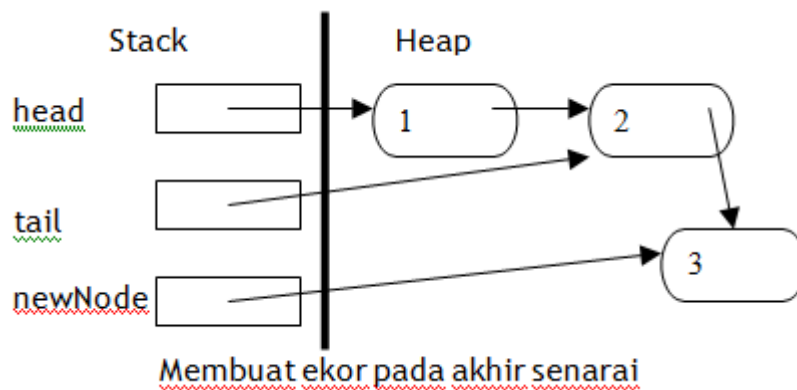
        push (&head, i);
    }
    // head == { 5, 4, 3, 2, 1 };
    return (head);
}

```

Dalam membuat tambahan node pada akhir senarai (list) dengan menggunakan perintah Push(), jika terjadi kesalahan maka urutan akan terbalik.

Membuat Ekor Pada Akhir Senarai

Menambah ekor pada senarai akan melibatkan penempatan node terakhir, dan kemudian merubahnya .next field dari NULL untuk menunjuk node baru seperti variabel tail dalam contoh yaitu menambah node 3 ke akhir daftar {1,2}



Untuk memasukkan atau menghapus node di dalam senarai, pointer akan dibutuhkan ke node sebelum posisi, sehingga akan dapat mengubah .next field. Agar dapat menambahkan node di akhir ekor pada data senarai {1, 2, 3, 4, 5}, dengan kesulitan node pertama pasti akan ditambah pada pointer kepala, tetapi semua node

yang lain dimasukkan sesudah node terakhir dengan menggunakan pointer ekor. Untuk menghubungkan dua hal tersebut adalah dengan menggunakan dua fungsi yang berbeda pada setiap permasalahan.

Fungsi pertama adalah menambah node kepala, kemudian melakukan perulangan yang terpisah yang menggunakan pointer ekor untuk menambah semua node yang lain. Pointer ekor akan digunakan untuk menunjuk pada node terakhir, dan masing-masing node baru ditambah dengan `tail->next`.

Contoh :

```
struct node*
BuildWithSpecialCase
() { struct node* head
= NULL; struct node*
tail;
int i;
push (&head, 1);
tail = head;
for (i=2; i<6; i++) {
    push (&(tail->next), i);
    tail = tail->next;
}
return (head);
}
```

Membuat Referansi Lokal

Dengan menggunakan “reference pointer” lokal, pointer akan selalu menunjuk ke pointer terakhir dalam senarai.

Semua tambahan pada senarai dibuat dengan reference pointer.

Reference pointer tidak menunjuk ke pointer kepala, tetapi menunjuk ke .next field didalam node terakhir dalam senarai.

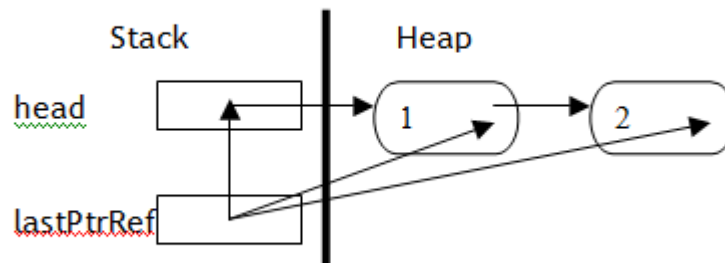
Contoh :

```
struct node*
BuildWithLocalRef() {
    struct node* head = NULL;
    struct node** lastPtrRef=&head;
    int i;

    for (i=2; i<6; i++) {
        Push (lastPtrRef, i);
        lastPtrRef=&((*lastPtrRef)->next);
    }
    return (head);
}
```

Cara kerja teknik ini adalah :

- Pada puncak putaran, lastPtrRef menunjuk pointer terakhir dalam senarai. Pada permulaannya menunjuk ke pointer kepala itu sendiri. Berikutnya menunjuk ke .next field di dalam node terakhir dalam senarai.
- Push(lastPtrRef); menambah node baru pada pointer akhir. Node baru menjadi node terakhir dalam senarai.
- lastPtrRef=&((*lastPtrRef)->next); lastPtrRef lanjut ke .next field dan .next field sekarang menjadi pointer terakhir dalam senarai.



Membuat ekor pada akhir senarai

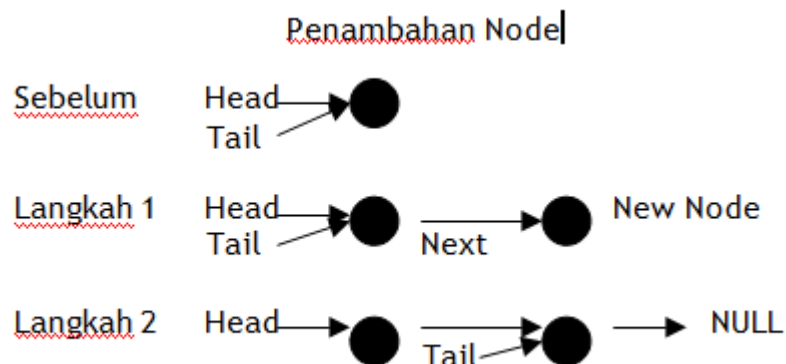
OPERASI DALAM LINKED LIST Menambah Node Baru

Untuk menambahkan sebuah node di dalam senarai, perlu didefinisikan sebuah kepala (head) dan ekor (tail). Pada awal senarai, posisi kepala dan ekor masih menjadi satu. Setelah ada tambahan node yang baru, maka posisinya berubah. Posisi kepala tetap pada awal senarai dan posisi ekor pada akhir senarai atau node yang baru. Bila ada tambahan node, maka posisi ekor akan bergeser sampai ke belakang dan akhir senarai ditunjukkan dengan NULL.

Contoh :

```
void SLList::AddANode()
{
    Tail->Next = new List;
    Tail=Tail->Next;

}
```



Menghapus Node

Contoh :

```
void SLList::DeleteANode(ListPtr corpse)
{
    ListPtr temp;
    if (corpse==Head) { temp=Head; Head=Head->Next;
                        delete temp;

    else if (corpse==Tail) { temp=Tail; Tail=Previous(Tail);
                            Tail->Next=NULL; delete temp;
```

<pre> }</pre>	<pre>temp=Previous(corpse); temp->Next=corpse->Next; delete corpse;</pre>
<pre>}</pre>	


```
        CurrentPtr=Head;
    }
```

CONTOH LATIHAN :

Buatlah program untuk memasukkan beberapa data dalam sebuah senarai (linked list), jika akan mengakhiri tekan n maka akan muncul semua node yang masuk ke dalam linked list tersebut.

//Program:link2

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
```

```
typedef struct node {
    int lkey;
    char name[10];
    struct node* next;
```

```
} TNODE;
```

```
TNODE *first, *last;
```

```
int LoadNode(TNODE *p);
void FreeNode(TNODE *p);
void PrintNode(TNODE *p);
```

```
void CreateList()
```

```
{
    TNODE *p;
    int n=sizeof(TNODE);

    first=last=0;
    for(;;)
    {
        if( (p=(TNODE*)malloc(n))==0 )
        {
            printf("\nmemori tidak cukup");
            break;
```

```

    }
    if(LoadNode(p)!=1)
    {
        FreeNode(p);
break;

```

```

    p->next=0;
    if (first==0)
first=last=p;
    else {
        last->next=p;
        last=p;
    }
}

```

```

int LoadNode(TNODE *p)
{
    char opt;
    printf("\nMasukkan node baru?");
    opt=getche(); opt=toupper(opt); if(opt!='N') {
        puts("\nMasukkan data untuk node:");
        printf("\nkey:\t");
        if (scanf("%d",&(p->lkey))!=1) return 0;
        printf("\nname:\t"); if (scanf("%s",p->name)!=1) return 0;
        return 1;
    }
    else
        return -1;
}

```

```

void FreeNode(TNODE *p) {
    free(p);
}

```

```

void ViewAllList()
{
    TNODE *p; p=first; while(p) {
        PrintNode(p);
        p=p->next;
    }
}

```

```
TNODE* FindNode(int key)
{
    TNODE *p; p=first; while(p) {
        if(p->lkey == key) return p;
        p=p->next;
    }
    return 0;
}

void PrintNode(TNODE *p)
{
    if(p) printf("\n%d\t%s",p->lkey,p->name);
}

TNODE* InsertBeforeFirst()
{
    TNODE *p;
    int n=sizeof(TNODE);
    if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1))
    {
        if (first==0) {
            p->next=0;
            first=last=p;
        }
    }
}
```

BAB VII

STACK (TUMPUKAN)

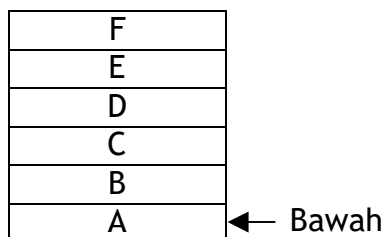
Stack merupakan bentuk khusus dari suatu struktur data, dimana node yang ditambahkan ke dalam list dan diambil dari list hanya pada kepalanya, atau dengan prinsip pengolahannya adalah last-in first-out (LIFO). Pada struktur ini hanya ada dua fungsi utama, yaitu push (memasukkan node ke dalam stack), dan pop (mengambil node dari stack).

PENGERTIAN TUMPUKAN

Secara sederhana tumpukan bisa diartikan sebagai kumpulan data yang seolah- olah diletakkan di atas data yang lain. Dalam suatu tumpukan akan dapat dilakukan operasi penambahan (penyisipan) dan pengambilan (penghapusan) data melalui ujung yang sama, ujung ini merupakan ujung atas tumpukan.

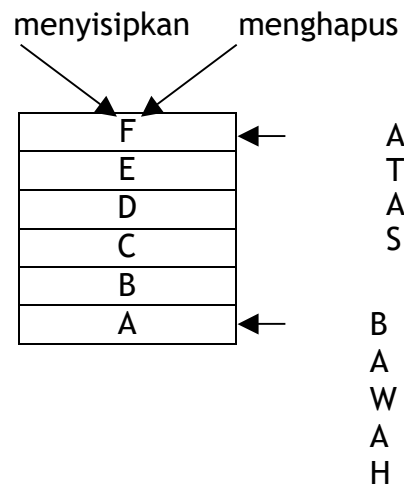
Contoh Kasus :

Terdapat dua buah kotak yang ditumpuk, kotak yang satu akan ditumpuk diatas kotak yang lainnya. Jika kemudian tumpukan 2 kotak tadi, ditambah kotak ketiga, keempat, kelima, dan seterusnya, maka akan diperoleh sebuah tumpukan kotak yang terdiri dari N kotak.



Dapat dilihat bahwa kotak B terletak diatas kotak A dan ada dibawah kotak C. Berdasarkan pada tumpukan tersebut dapat disimpulkan bahwa penambahan dan pengambilan sebuah kotak hanya dapat dilakukan dengan melewati satu ujung saja, yaitu bagian atas.

Kotak F adalah kotak paling atas sehingga jika ada kotak lain yang akan disisipkan maka kotak tersebut akan diletakkan pada posisi paling atas (diatas kotak F). Demikian juga pada saat pengambilan kotak, kotak F akan diambil pertama kali.



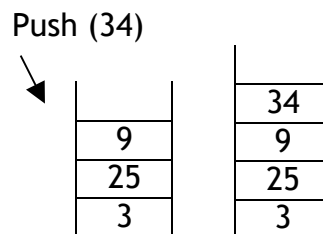
OPERASI PADA TUMPUKAN

Ada 2 operasi dasar yang bisa dilaksanakan pada sebuah tumpukan, yaitu :

- Menyisipkan data (push)
- Menghapus data (pop)

Operasi Push

Perintah push digunakan untuk memasukkan data ke dalam tumpukan.



Contoh :

```
void Push (NOD **T, char item)
{
    NOD *n;
    n=NodBaru (item);
    n->next=*T;
    *T=n;
}
```

Operasi Pop

Operasi pop adalah operasi untuk menghapus elemen yang terletak pada posisi paling atas dari sebuah tumpukan.

Untuk dapat mengetahui kosong tidaknya suatu tumpukan adalah dengan suatu fungsi yang menghasilkan suatu data bertipe boolean.

Contoh :

```
BOOL TumpukanKosong (NOD *T)
{
    return ((BOOL) (T==NULL));
}
```

Contoh :

```
char Pop (NOD **T)
{
    NOD *n; char item;
    if
        (!TumpukanKosong(*T)
        ) { P=*T;
        *T=(*T)->next;
        item=P->data;
        free(P);
        }
    return item;
}
```

PEMANFAATAN TUMPUKAN

Pemanfaatan tumpukan antara lain untuk menulis ungkapan dengan menggunakan notasi tertentu.

Contoh :

$(A + B) * (C - D)$

Tanda kurung selalu digunakan dalam penulisan ungkapan numeris untuk mengelompokkan bagian mana yang akan dikerjakan terlebih dahulu.

Dari contoh $(A + B)$ akan dikerjakan terlebih dahulu, kemudian baru $(C - D)$ dan terakhir hasilnya akan

dikalikan. $A + B * C - D$

$B * C$ akan dikerjakan terlebih dahulu, hasil yang didapat akan berbeda dengan hasil notasi dengan tanda kurung.

Notasi Infix Prefix

Cara penulisan ungkapan yaitu dengan menggunakan notasi infix, yang artinya operator ditulis diantara 2 operator.

Seorang ahli matematika bernama Jan Lukasiewicz mengembangkan suatu cara penulisan ungkapan numeris yang disebut prefix, yang artinya operator ditulis sebelum kedua operand yang akan disajikan.

Contoh :

Infix	Prefix
$A + B$	$+ A B$
$A + B - C$	$- + A B C$
$(A + B) * (C - D)$	$* + A B - C D$

Proses konversi dari infix ke prefix :

$$\begin{aligned}
 &= (A + B) * (C - D) \\
 &= [+ A B] * [- C D] \\
 &= * [+ A B] [- C D] \\
 &= * + A B - C D
 \end{aligned}$$

Notasi Infix Postfix

Cara penulisan ungkapan yaitu dengan menggunakan notasi postfix, yang artinya operator ditulis sesudah operand.

Contoh :

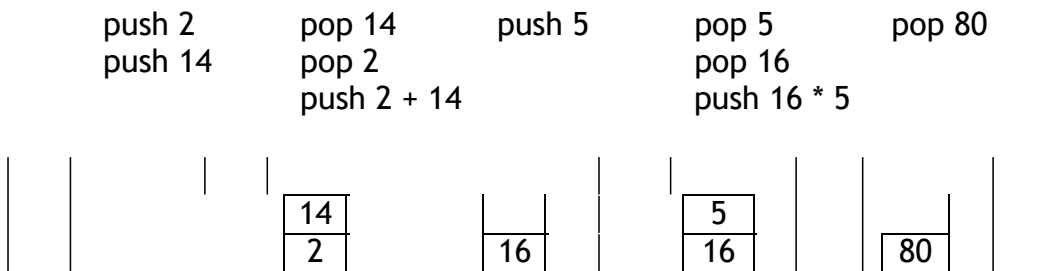
Infix	Postfix
$16 / 2$	$16 2 /$
$(2 + 14) * 5$	$2 14 + 5 *$
$2 + 14 * 5$	$2 14 5 * +$
$(6 - 2) * (5 + 4)$	$6 2 - 5 4 + *$

Proses konversi dari infix ke postfix :

$$\begin{aligned}
 &= (6 - 2) * (5 + 4) \\
 &= [6 2 -] * [5 4 +] \\
 &= [6 2 -] [5 4 +] * \\
 &= 6 2 - 5 4 + *
 \end{aligned}$$

Contoh :

Penggunaan notasi postfix dalam tumpukan, misal : $2\ 14 + 5 * = 80$



CONTO

H SOAL

: Soal 1

Buatlah program untuk memasukkan node baru ke dalam tumpukan.

//Program : stack1.cpp

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
```

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

```
struct nod {
    char data;
    struct nod *next;
};
```

```
typedef struct nod NOD;
```

```
BOOL TumpukanKosong(NOD *T) // Uji tumpukan kosong
{
    return ((BOOL)(T == NULL));
}
```

```
NOD *NodBaru (char item) //Ciptakan Node Baru
{
```

```

    NOD *n;

    n = (NOD*) malloc(sizeof(NOD));
    if(n != NULL) {
        n->data = item;
        n->next = NULL;
    }
    return n;
}

void CiptaTumpukan (NOD **T)
{
    *T = NULL; // Sediakan tumpukan Kosong
}

void Push(NOD **T, char item) // Push
{
    NOD *n;
    n = NodBaru(item);
    n->next = *T;
    *T = n;
}

char Pop(NOD **T) // Pop
{
    NOD *P; char item;
    if ( !
        TumpukanKo
        song(*T)) { P
        = *T;
        *T =
        (*T)-
        >next;
        item =
        P-
        >data;
        free(P);
    }
    return item;
}

void CetakTumpukan (NOD *T)
{
    NOD *p;

```

```
printf("T --> ");
for (p = T; p != NULL; p = p->next) {
    printf("[%c] --> ", p->data); }
printf("NULL\n");
}

int main()
{
    NOD *T; CiptaTumpukan(&T); Push(&T, 'I'); Push(&T, 'D');
    Push(&T, 'E'); CetakTumpukan(T); return 0;
}
```

Bila program tersebut

dijalankan maka : T ->

[E] -> [D] -> [I] -> NULL

Soal 2 :

Buatlah program untuk menampilkan kode pos (zip code) dari suatu negara bagian dan kota dan semua informasi tersebut dimasukkan ke dalam sebuah tumpukan. Apabila tidak ada keterangan yang dimasukkan berarti tumpukan kosong. Tekan q jika akan keluar.

//Program:stack2.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <io.h>
#define MAX_CITY 30
#define MAX_STATE 30
#define
MAX_ZIP
5 void
main
(void);
int is_empty (struct node *);
int push (struct node **);
int pop (struct node **);
void search (struct node **);
void free_nodes (struct node **pstack);

struct node {
    char zip_code[MAX_ZIP+1];
    char city[MAX_CITY]; char
    state[MAX_STATE]; struct node *link;
};

void main (void)
{
    struct node *pstack = NULL;
    int ok_so_far = 1, no_of_nodes = 0;
    while (ok_so_far == 1) {
        ok_so_far = push(&pstack);
        if (ok_so_far == 1)
            no_of_nodes ++;
        else if(ok_so_far == 0) {
            puts("\nAn unexpected error has occurred - terminating
            program");
            exit(1); //Abort program
        }
    }
    search (&pstack); //search linked list
    free_nodes(&pstack); //release memory back to OS when done
}

int push(struct node **pstack)
{
    struct node *new_ptr; //pointer for new struct
```

```
new_ptr = (struct node *) malloc(sizeof(struct node)); //memory for
new node if(new_ptr == (struct node *) NULL)//if malloc returns
NULL
{
    printf("ERROR! Unable to allocate memory - Abort\n");
    free(new_ptr);
    return (0); //return 0 so calling function knows an error occurred
}
else
{
    printf("\n\nEnter %d digit zip code or 'q' to quit>>", MAX_ZIP);
    gets(new_ptr->zip_code); //input zip code
    new_ptr->zip_code[MAX_ZIP] = '\0'; //NULL to 6th char in zip_code
    if (strcmp(new_ptr->zip_code, "q") != 0) {
        printf("\nEnter a less than %d character state name>>\n",
            MAX_STATE);
    }
}
```

```
        gets(new_ptr->state); //input state
        printf("\nEnter a less than %d character city
name>>\n", MAX_CITY);
        gets(new_ptr->city);
        //input city new_ptr-
        >link = *pstack;
        *pstack = new_ptr;
        return (1); //return 1 so calling func will continue to loop
    }
    else return (2);        //return 2 so calling func to stop looping
}
}
```

```
void search (struct node **pstack)
{
    struct node *ptemp;
    int test = 0;
    char ch, find[6];
    ptemp = *pstack;
    printf("\n\nEnter %d digit zip code to search for \nor 'e' to print
entire list>>", MAX_ZIP);
    gets(find); //input zip code
    find[MAX_ZIP] = '\0'; //assign NULL to 6th char in find array
    if (find[0] == 'E' || find[0] == 'e') //if user wants to view entire list
    {
        test = 1;
        while (test != 0) //while stack is not empty print
            test = pop (pstack); //info from stack and free nodes
    }
    else //otherwise search for zip code
    {
        while (test == 0 || ptemp != NULL) //while not found nor at
            the end of list
        {
            if (strcmp(ptemp->zip_code, find) == 0)
            {
                test = 1;
                printf("Zip Code: %s\n", ptemp->zip_code);
                printf("State: %s\n", ptemp->state);
                printf("City: %s\n\n", ptemp->city);
            }
            else if (ptemp == NULL)
            {
                printf("The zip code %s was not found.\n", find);
            }
        }
    }
}
```

```
        test = 1;
    }
    ptemp = ptemp->link;
}
puts ("\nType 'y' if you would you like to see the entire list");
puts ("or any other key to continue>>");
if (ch == 'y' || ch == 'Y')
{
    test = 1;
    while (test != 0)
        test = pop
            (pstack);
}
}
```

```
int pop (struct node **pstack)
{
    struct node *temp;
    if (is_empty(*pstack)== 1)
    {
        printf("\nStack is now empty");
        return(0);
    }
    else
    {
        temp = *pstack;
        printf("Zip Code: %s\n", temp->zip_code);
        printf("State: %s\n", temp->state);
        printf("City: %s\n\n", temp->city);
        *pstack = (*pstack)->link;
        free(temp);
        return(1);
    }
}
```



```
int is_empty (struct node *stack)    //test if stack points to NULL
{
    if (stack == NULL)
        return(1);                //if stack does point to NULL
    return 1 or true return(0);      //othrewise stack is
    not empty
}

void free_nodes (struct node **pstack)
{
    struct node *temp;              //temp pointer used for
    free()ing memory while (*pstack != NULL)
    {
        temp = *pstack;
        *pstack = (*pstack)->link;
        free(temp);    //release popped node's memory back to Operating
        System
    }
}
```

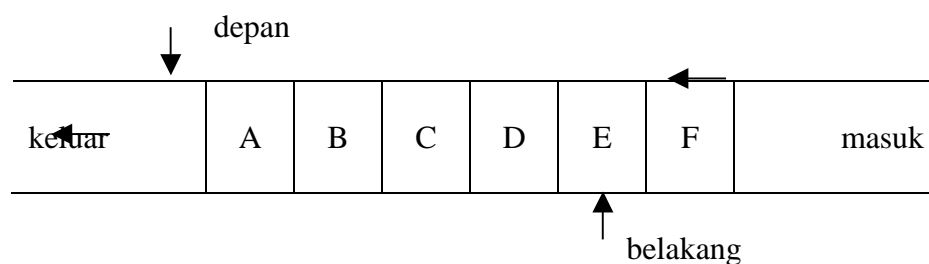
BAB VIII QUEUE (ANTRIAN)

Queue (Antrian) adalah suatu kumpulan data yang penambahan elemennya hanya bisa dilakukan pada suatu ujung (disebut dengan sisi belakang atau rear), dan penghapusan atau pengambilan elemen dilakukan lewat ujung yang lain (disebut dengan sisi depan atau front)

Jika pada tumpukan dikenal dengan menggunakan prinsip LIFO (Last In First Out), maka pada antrian prinsip yang digunakan adalah FIFO (First In First Out).

IMPLEMENTASI ANTRIAN DENGAN ARRAY

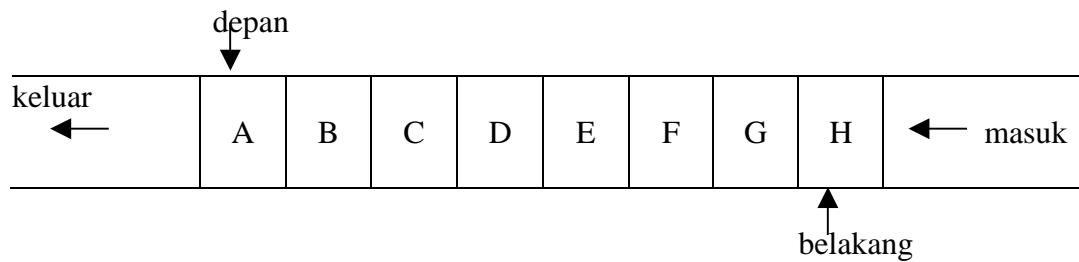
Karena antrian merupakan suatu kumpulan data, maka tipe data yang sesuai untuk menyajikan antrian adalah menggunakan array atau list (senarai berantai).



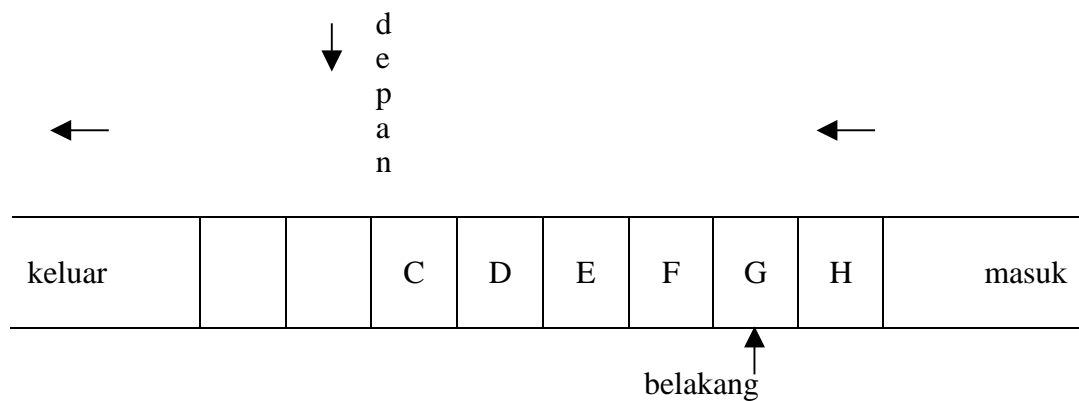
Antrian tersebut berisi 6 elemen, yaitu A, B, C, D, E, dan F. Elemen A terletak dibagian depan antrian dan elemen F terletak di bagian belakang antrian.

Jika terdapat elemen baru yang akan masuk, maka elemen tersebut akan diletakkan disebelah kanan F.

Jika ada elemen yang akan dihapus, maka A akan dihapus terlebih dahulu.



Antrian dengan penambahan elemen baru, yaitu G dan H.



Antrian dengan penghapusan elemen antrian, yaitu A dan B.

Seperti dalam tumpukan atau stack, maka di dalam antrian juga dikenal dua operasi dasar yaitu menambah elemen baru yang akan diletakkan di bagian belakang antrian dan menghapus elemen yang terletak di bagian depan antrian. Selain itu juga harus dilihat kondisi antrian mempunyai isi atau masih kosong.

Contoh :

```
#define MAXN 6
typedef enum {NOT_OK, OK} Tboolean;
typedef struct {
    Titem array[MAXN];
    int first;
    int last;
    int number_of_items;
} Tqueue;
```

Elemen antrian dinyatakan dalam tipe integer yang semuanya terdapat dalam struktur. Variabel first menunjukkan posisi elemen pertama dalam array, dan variabel last menunjukkan posisi elemen terakhir dalam array.

Untuk menambah elemen baru dan mengambil elemen dari antrian diperlukan deklarasi.

Contoh :

```
void initialize_queue (Tqueue
    *Pqueue) { Pqueue-
    >first=0;
    Pqueue->last=-1;
    Pqueue->number_of_items=0;
}
Tboolean enqueue (Tqueue *Pqueue, Titem item) {
    if (Pqueue->number_of_items >= MAXN)
        return (NOT_OK);
    else {

    }
}
```



```
Tboolean dequeue (Tqueue *Pqueue, Titem *Pitem) {  
    if (Pqueue->  
        >number_of_items==0)  
        return  
        (NOT_O  
         K);  
    else {  
        *Pitem=Pqueue->array[Pqueue->first++];  
        if (Pqueue->first > MAXN - 1)  
            Pqueue->first=0;  
        Pqueue->number_of_items--;  
        return (OK);  
    }  
}
```

Kondisi awal sebuah antrian dapat digambarkan sebagai berikut.

Antrian

6
5
4
3
2
1

first = 1 last = 0

Pada kondisi awal ini antrian terdiri dari last dibuat = 0 dan first dibuat = 1. Antrian dikatakan kosong jika $last < first$. Banyaknya elemen yang terdapat dalam antrian dinyatakan sebagai $last - first + 1$.

Antrian

6	
5	
4	D
3	C
2	B
1	A

last = 4

first = 1

Dengan $MAXN = 6$ antrian telah terisi empat buah data yaitu A, B, C, dan D. Kondisi first = 1 dan last = 4.

Antrian

6	
5	
4	D
3	C
2	
1	

last = 4

first = 3

Pada antrian dilakukan penghapusan dua buah data yaitu A dan B. Sehingga kondisi first = 3 dan last = 4.

Antrian

6	F
5	E
4	D
3	C
2	
1	

1

last = 6

first = 3

Pada antrian dilakukan penambahan dua buah data yaitu E dan F. Elemen E akan diletakkan setelah D dan elemen F akan diletakkan setelah E. Sehingga kondisi first = 3 dan last = 6.

Dapat diperoleh jumlah elemen yaitu $6 - 3 + 1 = 4$. Dengan pembatasan data maksimal 6 elemen akan terdapat sisa 2 elemen. Jika pada antrian akan

ditambahkan elemen yang baru, misal G, elemen G akan diletakkan setelah elemen F. Hal ini akan menyebabkan elemen yang baru tersebut tidak dapat masuk ($MAXN = 6$), meskipun masih tersisa 2 buah elemen yang kosong.

IMPLEMENTASI ANTRIAN DENGAN POINTER

Pada prinsipnya, antrian dengan pointer akan sama dengan antrian yang menggunakan array. Penambahan akan selalu dilakukan di belakang antrian dan penghapusan akan selalu dilakukan pada elemen dengan posisi paling depan. Antrian sebenarnya merupakan bentuk khusus dari suatu senarai berantai (linked list).

Pada antrian bisa digunakan dua variabel yang menyimpan posisi elemen paling depan dan elemen paling belakang. Jika menggunakan senarai berantai maka dengan dua pointer yang menunjuk elemen kepala (paling depan) dan elemen ekor (paling belakang) dapat dibentuk antrian.

Contoh :

```
struct queueNode {
    char data;
    struct queueNode * nextPtr;
};
typedef struct queueNode QUEUENODE;
typedef QUEUENODE
*QUEUENODEPTR; QUEUENODEPTR
headPtr = NULL, tailPtr = NULL;
```

Contoh :

Untuk menambah elemen baru yang selalu diletakkan di belakang senarai.

```
void enqueue (QUEUENODEPTR *headPtr, QUEUENODEPTR
*tailPtr, char value) {
    QUEUENODEPTR newPtr = malloc (sizeof(QUEUENODE));
    if (newPtr != NULL) { newPtr->data=value; newPtr-
        >nextPtr=NULL; if (isEmpty(*headPtr))
            *headPtr=newPtr;
        else
            (*tailPtr)->nextPtr=newPtr;
    }
    else
        return;
}
```

```
*tailPtr=newPtr; printf("%c not inserted. No memory available. \n", value);
```

Untuk menghapus akan dilakukan pemeriksaan terlebih dahulu antrian dalam keadaan kosong atau tidak.

```
int isEmpty (QUEUE_NODEPTR headPtr) {  
    return headPtr==NULL;  
}
```

CONTOH SOAL :

Soal 1

Buatlah program dalam bentuk menu untuk mengimplementasikan antrian. Menu tersebut berisi memasukkan data, menghapus data, menampilkan data, dan keluar

//Program : queue1

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

```
# include <conio.h>
```

```
class Linked_list_Queue
```

```
{
```

```
    private:
```

```
    struct node {
```

```
        int data;
```

```
        node *next;
```

```
    }; node *rear; node *entry; node *print; node *front;
```

```
    public: Linked_list_Queue( );
```

```
    void Delete( ); void Insert( ); void print_list( );
```

```
    void show_working( );
```

```
};
```

```
Linked_list_Queue::Linked_list_Queue ( )
```

```
{
```

```
    rear=NULL;
```

```
    front=NULL;
```

```
}
```

```
void Linked_list_Queue::Insert( )
```

```
{
```

```
    int num;
```

```
    cout<<"\n\n\n\n\n\tMasukkan angka dalam Queue : ";
```

```
    cin>>num; entry=new node; if(rear==NULL)
```

```
    {
```

```
        entry->data=num;
```

```
        entry->next=NULL;
```

```
        rear=entry;
```

```
        front=rear;
```

```
    }
```

```
    else
```

```
    {
```

```
e      ntry;  
n  
t  
r  
y  
-  
>  
d  
a  
t  
a  
=  
n  
u  
m  
;  
  
e  
n  
t  
r  
y  
-  
>  
n  
e  
x  
t  
=  
N  
U  
L  
L  
;  
  
r  
e  
a  
r  
-  
>  
n  
e  
x  
t  
=  
e
```

```
        rear=entry;
    }

    cout<<"\n\n\t *** "<<num<<" sudah masuk dalam Queue."<<endl;
    cout<<"\n\n\n\t\t Pres any key to return to Menu. ";
    getch( );
}

void Linked_list_Queue::Delete( )
{
    if(front==NULL)
        cout<<"\n\n\n\t *** Error : Queue is empty. \n"<<endl;
    else
    {
        int deleted_element=front->data;
        node *temp;
        temp=front;
        front=front->next;
        delete temp;
        cout<<"\n\n\n\t *** "<<deleted_element<<" dihapus dari Queue."<<endl;
    }

    cout<<"\n\n\n\t\t Pres any key to return to Menu. ";
    getch( );
}

void Linked_list_Queue::print_list( )
{
    print=front;
    if(print!=NULL)
        cout<<"\n\n\n\n\n\t Angka-angka yang ada dalam Queue adalah :
\n"<<endl;
    else
        cout<<"\n\n\n\n\n\t *** Tidak ada yang ditampilkan. "<<endl;
    while(print!=NULL)
    {
        cout<<"\t "<<print->data<<endl;
        print=print->next;
    }
    cout<<"\n\n\n\t\t Pres any key to return to Menu. ";
    getch( );
}
```

```
void Linked_list_Queue ::show_working( )
{
    char Key=NULL;
    do
    {
        clrscr( );
        gotoxy(5,
        5);
        gotoxy(10
        ,8);
        cout<<"Pilih salah satu menu :"<<endl;
        gotoxy(15,10);
        cout<<"- Press 'I' to Masukkan data dalam Queue"<<endl;
        gotoxy(15,12);
        cout<<"- Press 'D' to Hapus data dari Queue"<<endl;
        gotoxy(15,14);
        cout<<"- Press 'P' to Tampilkan data dari Queue"<<endl;
        gotoxy(15,16);
```

```
cout<<"- Press \'E\' to  
  
Exit"<<endl; Input:  
  
gotoxy(10,20);  
cout<<"          ";  
gotoxy(10,20);  
cout<<"Masukkan Pilihan  
: "; Key=getche( );  
if(int(Key)==27 || Key=='e' || Key=='E')  
    break;  
else if(Key=='i' ||  
        Key=='I') Insert(  
    );  
else if(Key=='d' ||  
        Key=='D')  
    Delete( );  
else if(Key=='p' || Key=='P')  
    print_list( );  
else  
    goto Input;  
}
```

```
        while(1);  
    }  
  
    int main( )  
    {  
        Linked_list_Q  
        ueue obj;  
        obj.show_wor  
        king( ); return  
        0;  
    }
```

SOAL LATIHAN :

Buatlah program untuk memasukkan dan mengeluarkan data dalam antrian. Contoh ada di file : queue2.exe

Save dengan nama file : qu1_nim (4 digit nim terakhir)

BAB IX TREE (POHON)

Struktur pada tree (pohon) tidak linear seperti pada struktur linked list, stack, dan queue. Setiap node pada tree mempunyai tingkatan, yaitu orang tua (parent) dan anak (child). Struktur ini sebenarnya merupakan bentuk khusus dari struktur tree yang lebih umum, setiap orang tua hanya memiliki dua anak sehingga disebut pohon biner (binary tree), yaitu anak kiri dan anak kanan.

ISTILAH DASAR

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree dapat didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut root dan node, disebut sub tree/sub pohon atau cabang.

Sehingga secara sederhana pohon bisa didefinisikan sebagai kumpulan elemen yang salah satu elemennya disebut dengan akar (root) dan elemen yang lainnya

(simpul), terpecah menjadi sejumlah himpunan yang saling tidak berhubungan

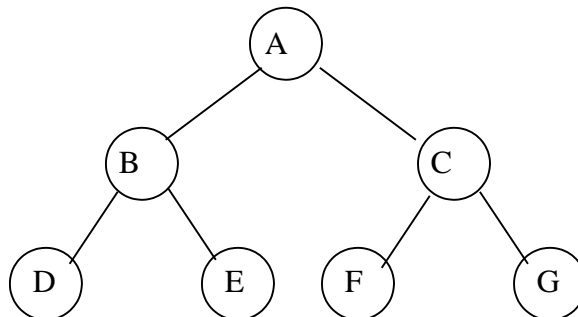
satu dengan yang

lainnya.

- Predecessor : node yang berada di atas node tertentu
- Successor : node yang dibawah node tertentu
- Ancestor : seluruh node yang terletak sebelum node tertentu dan terletak sesudah pada jalur yang sama

- Descendant : seluruh node yang terletak sesudah node tertentu dan terletak sesudah pada jalur yang sama
- Parent : predecessor satu level diatas suatu node
- Child : successor satu level dibawah suatu node
- Sibling : node-node yang memiliki parent yang sama dengan suatu node
- Subtree : bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut
- Size : banyaknya node dalam suatu tree
- Height : banyaknya tingkatan/level dalam suatu tree
- Root : satu-satunya node khusus dalam tree yang tidak mempunyai predecessor
- Leaf : node-node dalam tree yang tidak memiliki successor
- Degree : banyaknya child yang dimiliki suatu node

Contoh :



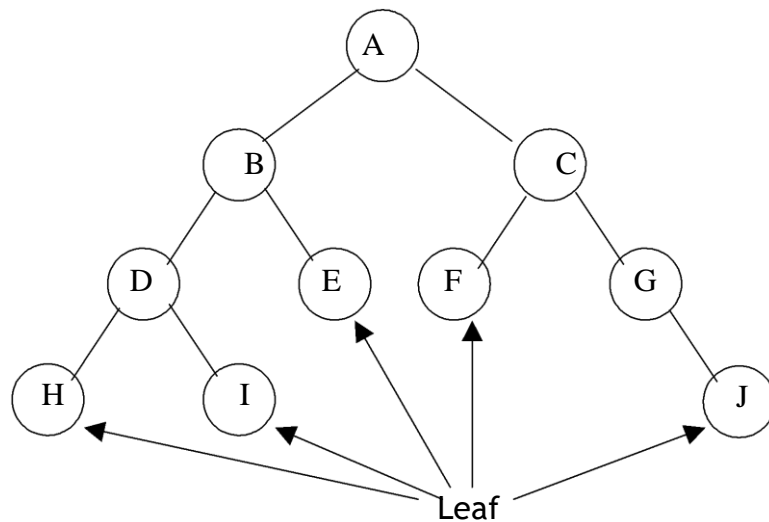
Keterangan :

- Ancestor : C, A
- Descendant : C, G
- Parent : B
- Child : B, C
- Sibling : F, G
- Size : 7
- Height : 3
- Root : A
- Leaf : D, E, F, G
- Degree : 2

JENIS TREE

1. Binary Tree

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



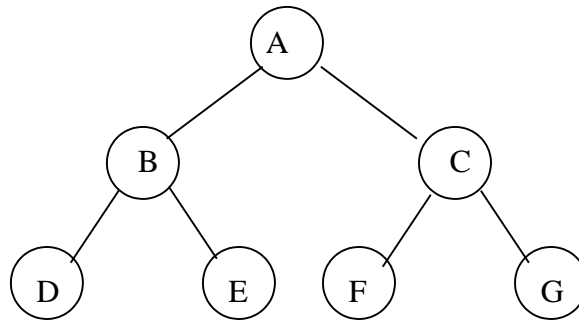
Keterangan :

- Left Child : B, D, H, ...
- Right Child : C, G, J, ...

Jenis Binary Tree

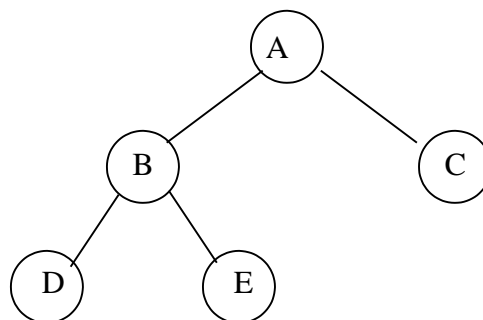
Full Binary Tree

Binary Tree yang tiap nodenya (kecuali leaf) memiliki dua child dan tiap sub tree harus mempunyai panjang path yang sama.



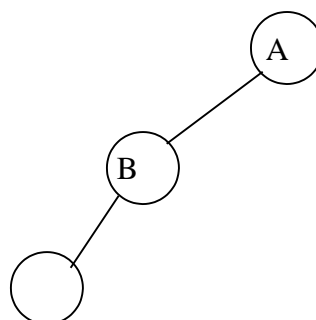
Complete Binary Tree

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



Skewed Binary Tree

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



D

Operasi 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 3 pilihan insert, yaitu : 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.
- Retrive : mengetahui isi dari node yang ditunjuk oleh pointer current. Tree tidak boleh kosong.
- DeleteSub : menghapus sebuah subtree (node beserta seluruh descendantnya) 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 lengthnya. Tree tidak boleh kosong.
- Traverse : mengunjungi seluruh node pada tree, masing-masing sekali. Hasilnya adalah urutan informasi secara linear yang tersimpan dalam tree. Ada tiga cara traverse : Pre Order, InOrder, dan PostOrder.

Langkah Traverse

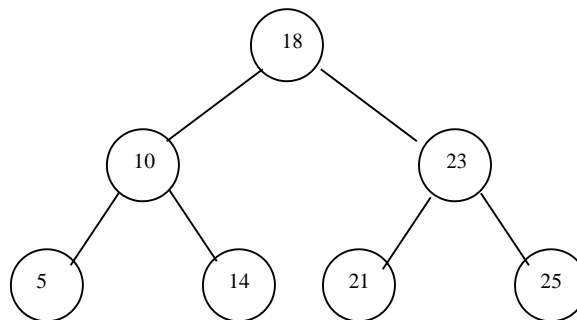
- PreOrder : cetak isi node yang dikunjungi, kunjungi Left Child, kunjungi Right Child.
- InOrder : kunjungi Left Child, cetak isi node yang dikunjungi, kunjungi Right Child.

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

2. Binary Search Tree

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

Contoh :



Operasi Binary Tree

Pada dasarnya sama dengan operasi pada binary tree, kecuali pada operasi insert, update, dan delete.

- Insert : dilakukan setelah ditemukan lokasi yang tepat, lokasi tidak ditentukan oleh user sendiri.
- Update : update akan berpengaruh pada posisi node tersebut selanjutnya. Setelah diupdate mengakibatkan tree tersebut bukan binary search tree lagi, maka harus dilakukan perubahan pada tree dengan melakukan rotasi supaya tetap menjadi binary search tree.
- Delete : akan mempengaruhi struktur dari tree.

3. AVL Tree

Adalah binary search tree yang memiliki perbedaan tingkat tinggi/level antara subtree kiri dan subtree kanan maksimal adalah 1. Dengan AVL Tree, waktu pencarian dan bentuk tree dapat dipersingkat dan disederhanakan.

Selain AVL Tree terdapat juga Height Balanced n tree, yaitu binary search tree yang memiliki perbedaan level antara subtree kiri dan subtree kanan maksimal adalah n. Sehingga AVL Tree adalah Height Balanced 1 Tree.

Simbol Bantu

Untuk mempermudah menyeimbangkan tree, maka digunakan simbol-simbol bantu.

- Minus (-) : digunakan apabila subtree kiri lebih panjang dari subtree kanan.
- Plus (+) : digunakan apabila subtree kanan lebih panjang dari subtree kiri.
- Nol (0) : digunakan apabila subtree kiri dan subtree kanan mempunyai height yang sama.

CONTOH SOAL :

Soal 1

Buatlah program untuk menampilkan node baru ke dalam pohon dengan menggunakan prosedur preorder, inorder, dan postorder.

```
//Program :tree.cpp
#include <stdio.h>
#include <malloc.h>
```

```
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 != NULL) {
            n->data = item;
            n->left = NULL;
            n->right = NULL;
        }
        return n;
    }

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

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

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

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

void preOrder(POKOK *T) {
    if(!PokokKosong(T)) { printf("%c ", T->data);
        preOrder(T->left); preOrder(T->right);
    }
}

void inOrder(POKOK *T) {
    if(!PokokKosong(T)) { inOrder(T->left); printf("%c ",
        T->data); inOrder(T->right);
    }
}

void postOrder(POKOK *T) {
    if(!PokokKosong(T)) { postOrder(T->left);
        postOrder(T->right); printf("%c ", T->data);
    }
}

int main()
{
    POKOK *kelapa; char buah; BinaPokok(&kelapa);
    TambahNod(&kelapa, buah = 'M');
    TambahNod(&kelapa->left, buah = 'E');
    PTambahNod(&kelapa->left->right, buah =
    'T'); TambahNod(&kelapa->right, buah =

```



```
'L'); TambahNod(&kelapa->right->right,  
buah = 'O'); TambahNod(&kelapa->right-  
>right->left, buah = 'D'); printf("Tampilan  
secara PreOrder: "); preOrder(kelapa);  
printf("\nTampilan secara InOrder: ");  
inOrder(kelapa);
```

```
printf("\nTampilan secara PreOrder: ");  
postOrder(kelapa);  
;  
printf("\n  
\n");  
return 0;
```

```
}
```

SOAL LATIHAN :

Buatlah program untuk menampilkan node baru ke dalam pohon dengan menggunakan prosedur preorder, inorder, dan postorder.

Sehingga akan didapatkan hasil :

Tampilan secara PreOrder : R
A S I T E Tampilan secara
InOrder : I S T A R E Tampilan
secara PostOrder : I T S A E R

Contoh ada di file : tree2.exe

Save dengan nama file : tre_nim (4 digit nim terakhir)