



MODUL PELATIHAN PEMROGRAMAN PEMBERDAYAAN UMAT BERKELANJUTAN

Struktur Data



أَظْلِبِ الْعِلْمَ مِنَ الْمَهْدِ إِلَى الْلَّهِ

"Tuntutlah ilmu sejak dari buaian sampai ke liang lahad"

Kata pengantar

Assalamu'alaikum warahmatullahi wabarakatuh

Puji syukur kehadirat Allah SWT karena atas izinnya kami berkesempatan untuk merevisi modul pelatihan ini. Shalawat serta salam kepada Nabi Muhammad SAW yang telah membawa kita dari zaman kebodohan menuju zaman terang-benderang yang penuh dengan ilmu pengetahuan.

Dulu kita menggunakan compiler Borland yang sintaksisnya agak asing jika dibandingkan compiler-compiler C standar yang banyak digunakan di dunia. Pada tahun 2021, PUB mulai beralih ke compiler C standar, yaitu GCC. Oleh karena itu modul Struktur Data peninggalan angkatan-angkatan sebelumnya ini kami revisi lagi agar tetap bisa digunakan.

Mungkin banyak sekali kekurangan di modul revisi ini. Salah satu kekurangannya adalah tidak adanya daftar isi. Ini disebabkan karena modul ini merupakan hasil gabungan dari modul lama yang ditulis dalam format dokumen Microsoft Word dan halaman-halaman baru yang ditulis dalam format Markdown. Selain itu redaksi bahasa, kebakuan kata, ejaan, serta tanda baca banyak yang belum diperbaiki karena terkendala oleh waktu.

Selama berbulan-bulan kami terus mencari tahu bagaimana cara membuat modul yang bisa ditampilkan dalam 4 bentuk, yaitu web berbasis React, mobile berbasis Flutter, desktop berbasis Electron, dan PDF seperti biasanya, mencari tahu bagaimana para developer menulis dokumentasi karya mereka. Akhirnya kami menemukan jawabannya, yaitu Markdown.

Oleh karena itu ke depannya semua modul akan ditulis dalam format Markdown supaya bisa ditampilkan dengan baik di website PUB dan super-app PUB Portal, dan tetap bisa dieksport ke PDF.

Kami akan terus berusaha untuk meningkatkan kualitas modul pelatihan di Program Beasiswa PUB, tentunya dengan dukungan dari semua pihak. Terima kasih kepada semuanya yang telah bersedia membantu dalam misi ini.

Wassalamu'alaikum warahmatullahi wabarakatuh

Bandung, 25 Januari 2022

KOORDINATOR DIVISI PENDIDIKAN
PEMBERDAYAAN UMAT BERKELANJUTAN

ttd.

ROMI KUSUMA BAKTI

Silabus sementara

Pertemuan	Topik pembahasan
1	Pengenalan struktur data, array, dan structure
2	Pointer
3	Linked list (insert, delete, dan print)
4	Linked list (update dan search)
5	Double linked list
6	Stack
7	Queue
8	<i>Ujian pertengahan</i>
9	Sorting (pengertian dan kegunaan)
10	Sorting (bubble dan selection)
11	Sorting (insertion dan counting)
12	Sorting (quick dan merge)
13	Tree
14	Graph
15	Graph (algoritma jalur terpendek)
16	<i>Ujian akhir</i>

Bobot nilai

Nilai	Persentase
Rata-rata tugas	10%
Rata-rata kuis	20%
Ujian pertengahan	30%
Ujian akhir	40%

Pendahuluan

Daftar isi

- Pendahuluan
 - Daftar isi
 - Persiapan
 - Apa saja yang harus disiapkan?
 - Apa IDE yang akan digunakan?
 - Jadi masih mungkinkah kita menggunakan Borland?
 - IDE apa saja yang bisa menggunakan compiler GCC?
 - Memperkenalkan Borland X
 - Struktur data
 - Mengenal struktur data
 - Kegunaan struktur data
 - Konsep dan definisi data
 - Tipe data
 - Tipe data primitif
 - Tipe data komposit

Persiapan

Apa saja yang harus disiapkan?

- Pengetahuan tentang algoritma
- Pengetahuan tentang pemrograman C
- Compiler C
- IDE C

Apa IDE yang akan digunakan?

Sebenarnya kami tidak pernah mempermasalahkan IDE yang digunakan. **Kesalahan besar** jika ada yang mengira Divisi Pendidikan mengganti IDE yang digunakan dari Borland ke Visual Studio Code, karena terkesan modern, tampilannya keren, dsb., ini mungkin yang ada di pikiran orang-orang awam yang tidak mau mencari tahu. Padahal yang sebenarnya kami lakukan adalah **mengganti compiler dari Borland Turbo ke GCC** yang sesuai standar.

Jadi masih mungkinkah kita menggunakan Borland?

Tidak, karena Borland hanya bisa menggunakan compiler Borland Turbo, kecuali jika ditemukan cara untuk mengganti compiler Borland ke GCC.

IDE apa saja yang bisa menggunakan compiler GCC?

- Eclipse
- Visual Studio Code
- Code::Blocks
- Dev-C++
- CLion
- CodeLite
- dan masih banyak lagi.

Memperkenalkan Borland X

Saat ini kami sedang mengembangkan software bernama Borland X, IDE C/C++ dengan compiler GCC yang dibangun menggunakan framework Electron yang ditulis dalam Typescript. Ini adalah project jangka panjang yang tidak mungkin selesai dalam waktu dekat. Saat ini Borland X masih dalam tahap pengembangan dan belum siap digunakan.

Repository: <https://github.com/romikusumabakti/borland-x>

Struktur data

Mengenal struktur data

Struktur data adalah cara tertentu untuk mengatur dan menyimpan data di komputer sedemikian rupa sehingga dapat digunakan secara efisien.

Sedangkan data adalah representasi dari fakta dunia nyata yang disimpan, direkam, atau direpresentasikan dalam bentuk tulisan, suara, gambar, sinyal, atau simbol.

Struktur data digunakan di hampir setiap program atau sistem yang telah dikembangkan. Selain itu, struktur data berada di bawah dasar-dasar Ilmu Komputer dan RPL. Ini adalah topik utama dalam pertanyaan wawancara (interview). Oleh karena itu sebagai programmer, kita harus memiliki pengetahuan yang baik tentang struktur data.

Tipe struktur data umum meliputi array, file, record, tabel, pohon, dan sebagainya. Setiap struktur data dirancang untuk mengatur data agar sesuai dengan tujuan tertentu sehingga bisa diakses dan dikerjakan dengan baik.

Kegunaan struktur data

Struktur data digunakan untuk meningkatkan efisiensi penggunaan memori pada saat program komputer sedang bekerja. Penggunaan struktur data yang tepat pada pemrograman dapat membuat algoritma menjadi lebih mudah dan membuat program lebih efisien.

Meningkatkan efisiensi merupakan tujuan utama pengaplikasian struktur data. Dengan struktur data, proses reservasi memori yang tidak perlu akan diminimalisasi. Selain itu struktur data juga menjamin kemudahan pemahaman algoritma.

Konsep dan definisi data

Data adalah fakta atau kenyataan tercatat mengenai suatu objek. Data menyiratkan suatu nilai, nilai bisa dalam bentuk konstanta atau variabel. Data ada yang bersifat kuantitatif dan kualitatif. Data kuantitatif berupa angka, misalnya nilai UAS. Data kualitatif berupa non-angka, misalnya nama.

Tipe data

Tipe data dibagi menjadi dua, yaitu tipe data primitif dan tipe data komposit.

Tipe data primitif

Tipe data primitif adalah tipe data yang hanya dapat menampung satu nilai dalam satu variabel.

Macam-macam tipe data primitif:

1. Integer

Integer (bilangan bulat) merupakan nilai bilangan bulat baik dalam bentuk desimal maupun heksadesimal.

Penggolongan tipe data integer:

Tipe data	Ukuran	Tempat rentang nilai
byte	1 byte	0 s/d 255
short int	1 byte	-128 s/d 127
int	2 byte	-32768 s/d 32767
word	2 byte	0 s/d 65535
long int	4 byte	2147483648 s/d 2147483647

Contoh:

```
int a;  
// Mendeklarasikan variabel bertipe integer yang bernama "a".
```

```
int a = 10;  
// Mendeklarasikan variabel bertipe integer yang bernama "a".  
// Kemudian menginisialisasinya dengan bilangan 10
```

2. Floating point

Floating point adalah bilangan pecahan.

Penggolongan tipe data floating point:

Tipe data	Ukuran	Tempat rentang nilai
real	6 byte	2.9×10^{-39} s/d 1.7×10^{38}
single	4 byte	2.9×10^{-39} s/d 1.7×10^{38}
double	8 byte	5.0×10^{-324} s/d 1.7×10^{308}
extended	10 byte	3.4×10^{-4932} s/d 1.1×10^{4932}
comp	8 byte	3.4×10^{-4932} s/d 1.1×10^{4932}

3. Boolean

Tipe data ini digunakan untuk pengambilan keputusan dalam operasi logika. Terdiri dari `true` disimbolkan 'T' dan `false` yang disimbolkan 'F'. Ketika kita ingin mendapatkan hasil yang valid/pasti, kita menggunakan tipe data boolean untuk memperoleh keputusan dalam suatu penyelesaian yang pasti.

4. Character

Character adalah karakter tunggal yang didefinisikan dengan diawali dan diakhiri dengan tanda ' (petik tunggal). Tipe data char hanya membutuhkan 1 byte memori (1 byte = 8 bit). Selain itu, variabel bertipe data char juga bisa diisi dengan urutan karakter ASCII.

Nilai-nilai yang termasuk karakter adalah:

- Karakter huruf: 'a'..'z', 'A'..'Z'
- Karakter angka: '0'..'9'
- Karakter tanda baca : titik, koma, titik koma, titik dua dan sebagainya
- Karakter khusus: \$, %, #, @ dan sebagainya. Contoh:

```
char huruf = 'a';
```

```
char angka = '7';
```

Tipe data komposit

Tipe data komposit adalah tipe data di mana satu variabel dapat menyimpan lebih dari satu nilai data.

Macam-macam tipe data komposit:

1. Array

Contoh: array integer, array karakter (string), dll.

2. Structure

Contoh: struct mahasiswa, struct buku, dll.

Array

Pengertian array

Secara bahasa, array berarti himpunan, susunan, atau jajaran. Array juga sering disebut larik dalam bahasa Indonesia.

Array adalah struktur data yang terdiri dari kumpulan elemen berupa nilai yang masing-masing diidentifikasi oleh indeks atau key.

Array dapat menyimpan lebih dari 1 data dengan tipe yang sama. Misalnya sekumpulan bilangan, sekumpulan karakter, dan sebagainya.

Membuat array

```
tipe_data nama_variabel[ukuran];
```

ukuran adalah berapa banyak data yang dapat disimpan oleh variabel itu. Ukuran array tidak dapat diubah setelah ditentukan.

Contoh:

```
int sekumpulan_bilangan[10];
```

Variabel di atas dapat menyimpan 10 data dengan tipe integer.

```
char sekumpulan_huruf[26];
```

Variabel di atas dapat menyimpan 26 data dengan tipe karakter.

Inisialisasi array

Nilai-nilai dalam array disebut dengan “elemen”.

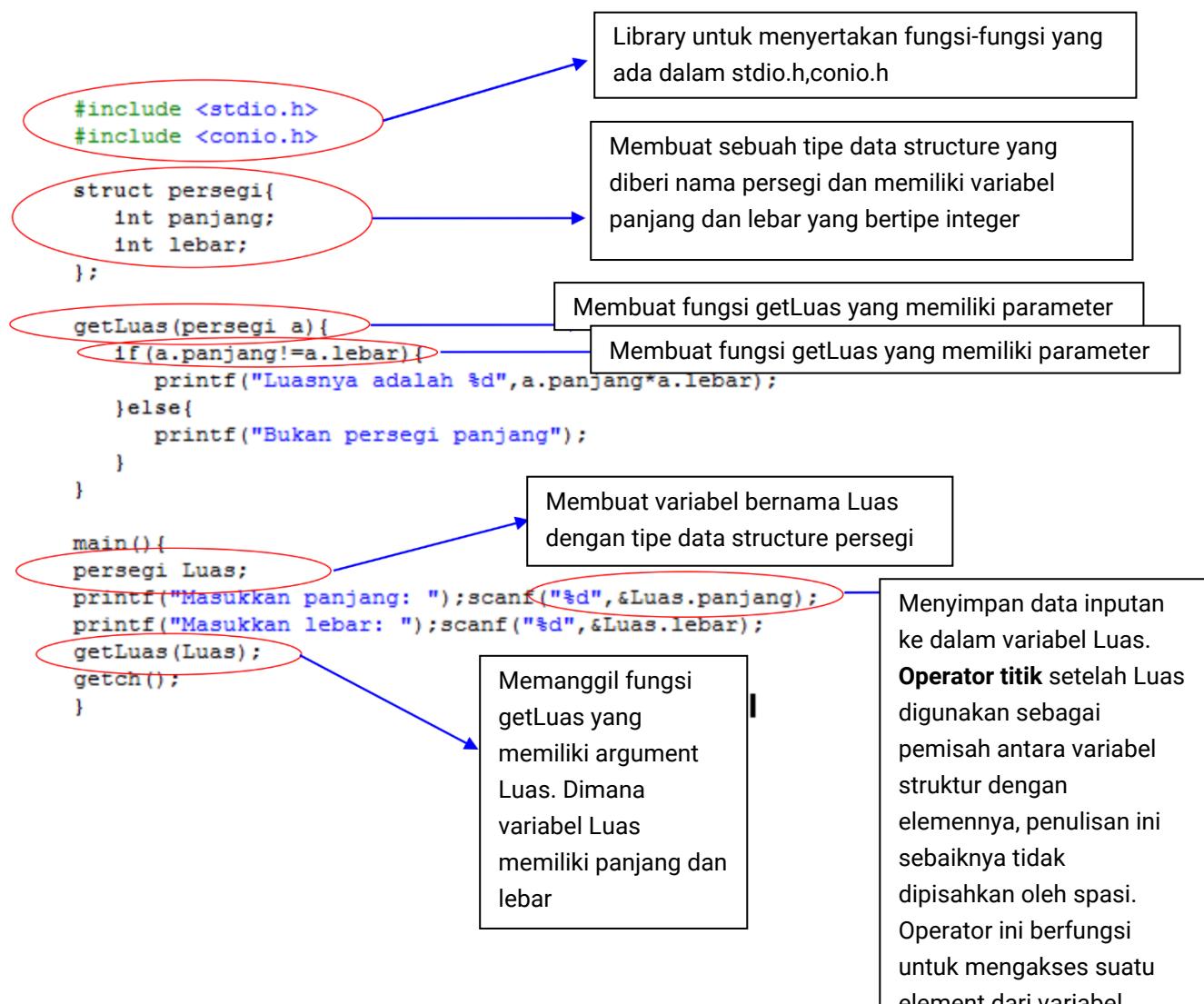
```
tipe_data nama_variabel[ukuran] = {elemen 1, elemen 2, ...};
```

Contoh:

```
int bilangan_prima[7] = {2, 3, 5, 7, 11, 13, 17};
```

```
char huruf_vokal[5] = {'a', 'i', 'u', 'e', 'o'};
```

Contoh 1:



Output:

```

Masukkan panjang: 6
Masukkan lebar: 4
Luasnya adalah 24

```

Contoh 2:

```
// membuat struct
struct Mahasiswa {
    char *nama;
    char *alamat;
    int umur;
};

void main(){
    // menggunakan struct
    struct Mahasiswa mhs1, mhs2;

    // mengisi nilai ke struct
    mhs1.nama = "Loli";
    mhs1.alamat = "Bandung";
    mhs1.umur = 22;

    mhs2.nama = "Bambang";
    mhs2.alamat = "Surabaya";
    mhs2.umur = 23;

    // mencetak isi struct
    printf("==Mahasiswa 1 ==\n");
    printf("Nama: %s\n", mhs1.nama);
    printf("Alamat: %s\n", mhs1.alamat);
    printf("Umur: %d\n", mhs1.umur);

    printf("==Mahasiswa 2 ==\n");
    printf("Nama: %s\n", mhs2.nama);
    printf("Alamat: %s\n", mhs2.alamat);
    printf("Umur: %d\n", mhs2.umur);
}
```

Output:

```
c:\Pelatihan C\struct.exe
==Mahasiswa 1 ==
Nama: Loli
Alamat: Bandung
Umur: 22
==Mahasiswa 2 ==
Nama: Bambang
Alamat: Surabaya
Umur: 23
```

Konvensi penamaan structure

Berikut ini konvensi penamaan yang umum digunakan di dunia pemrograman.

Pemformatan	Nama
katakata	flatcase
KATAKATA	UPPERFLATCASE
kataKata	camelCase atau lowerCamelCase
KataKata	PascalCase atau UpperPascalCase
kata_kata	snake_case
kata-kata	kebab-case

Karena belum ada aturan pasti mengenai konvensi penamaan untuk structure, jadi kami memutuskan untuk menjadikan PascalCase sebagai standar konvensi penamaan structure di pelatihan, jadi cara penamaannya adalah "[NamaStructure](#)" bukan "[nama_structure](#)".

Sedangkan untuk membuat variabel, kita tetap menggunakan snake_case seperti biasanya.

Contoh:

```
#include <stdio.h>

struct ProgramStudi
{
    char *jenjang;
    char *jurusan;
};

void main()
{
    struct ProgramStudi program_studi_1;
    program_studi_1.jenjang = "D3";
    program_studi_1.jurusan = "Manajemen Informatika";

    struct ProgramStudi program_studi_2;
    program_studi_2.jenjang = "S1";
    program_studi_2.jurusan = "Teknik Informatika";

    printf("%s %s\n", program_studi_1.jenjang, program_studi_1.jurusan);
    printf("%s %s", program_studi_2.jenjang, program_studi_2.jurusan);
}
```

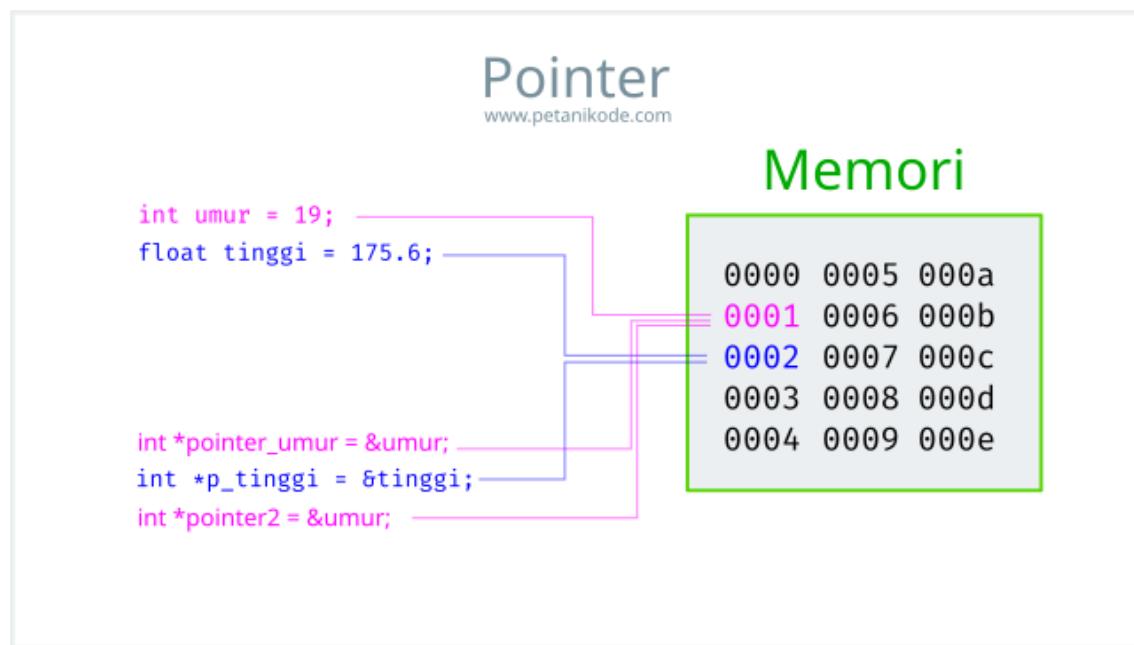
Pointer

Pointer adalah tipe data yang berisi alamat lokasi memori dari sebuah variabel. Sebelumnya variabel adalah suatu wadah atau tempungan yang digunakan untuk menyimpan suatu nilai. Dalam deklarasi variabel, variabel tersebut memesan lokasi (tempat) dalam memori secara acak.

Variabel pointer sering dikatakan sebagai variabel yang menunjuk ke variabel lain. Sebenarnya variabel pointer berisi alamat dari variabel lain (yang dikatakan ditunjuk oleh pointer). Jadi pointer tidak berisi nilai/data variabel, melainkan berisi alamat lokasi memori dari variabel itu.

Memori komputer ibarat sebuah lemari besar, variabel ibarat laci-laci yang ada pada lemari itu. Data disimpan di dalam setiap laci. Setiap laci tentunya memiliki nomor agar mudah diakses. Nomor laci inilah yang disebut alamat lokasi memori.

Nomor laci juga bisa kita simpan ke dalam laci, misalnya dengan mencatatnya di kertas kemudian menyimpannya ke dalam laci tertentu. Ini sama seperti kita menyimpan alamat lokasi memori variabel ke dalam variabel lain.



Membuat variabel pointer

Variabel pointer dideklarasikan dalam bentuk sebagai berikut:

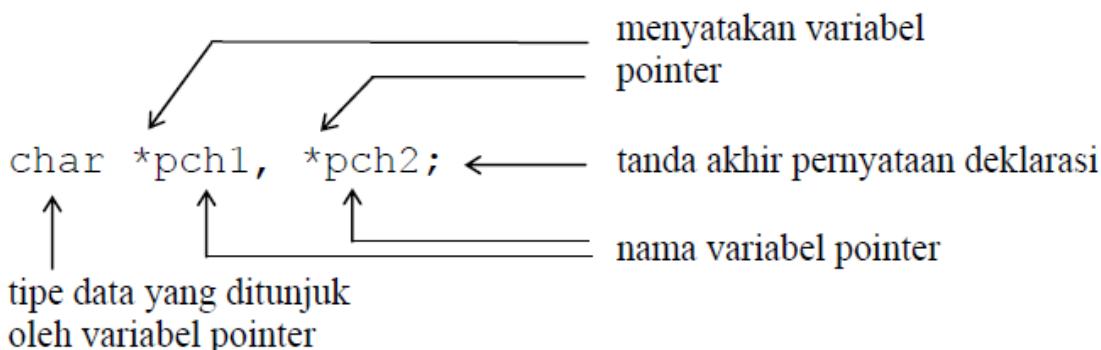
```
tipe_data *nama_variabel;
```

Variabel pointer mempunyai tanda * sebelum nama variabel. Tipe data bisa menggunakan apa saja, tetapi pointer tersebut harus menunjuk ke variabel bertipe sama.

Contoh:

```
int *px;  
char *pch1,*pch2;
```

Contoh pertama menyatakan bahwa px adalah variabel pointer yang menunjuk ke suatu data bertipe integer. Sedangkan contoh kedua pch1 dan pch2 menunjuk ke data bertipe char. Contoh deklarasi variabel dinyatakan seperti ini:



Membuat variabel pointer yang menunjuk variabel lain

Agar variabel pointer dapat menunjuk ke variabel lain, pointer harus diisi dengan alamat dari variabel yang akan ditunjuk. Untuk menyatakan alamat dari suatu variabel, gunakan operator `&` (operator alamat, bersifat unary) sebelum nama variabel. Misalnya jika variabel `x` sudah dideklarasikan maka untuk mendapatkan alamatnya adalah seperti ini:

`&x`

yang berarti “alamat dari variabel `x`”. Untuk memasukkan (assign) alamat `x` ke variabel pointer, misalnya `px` (yang dideklarasikan sebagai pointer yang menunjuk ke data bertipe sama dengan variabel `x`) yaitu:

`px = &x;`

Pernyataan di atas berarti bahwa `px` diberi nilai berupa alamat dari variabel `x`. Setelah pernyataan tersebut dieksekusi barulah dapat dikatakan bahwa `px` menunjuk ke variabel `x`.

Mengakses isi suatu variabel melalui pointer

Jika variabel sudah ditunjuk oleh pointer, variabel yang ditunjuk oleh pointer tersebut dapat diakses melalui variabel itu sendiri (pengaksesan langsung). Pengaksesan tak langsung dilakukan dengan menggunakan operator indirection (tak langsung) berupa simbol `*` (bersifat unary). Contoh penerapan operator `*` seperti ini:

`*px;`

Yang menyatakan “isi atau nilai variabel/data yang ditunjuk oleh pointer `px`”. Sebagai contoh:

`px = &x;`

y = *px;

y yang berisi nilai yang sama dengan nilai x. Untuk jelasnya perhatikan contohnya:

```
#include <stdio.h>

int main(){
    // x & y bertipe int
    int y, x = 99;

    //variabel pointer yang menunjuk ke data yang bertipe int
    int *px;

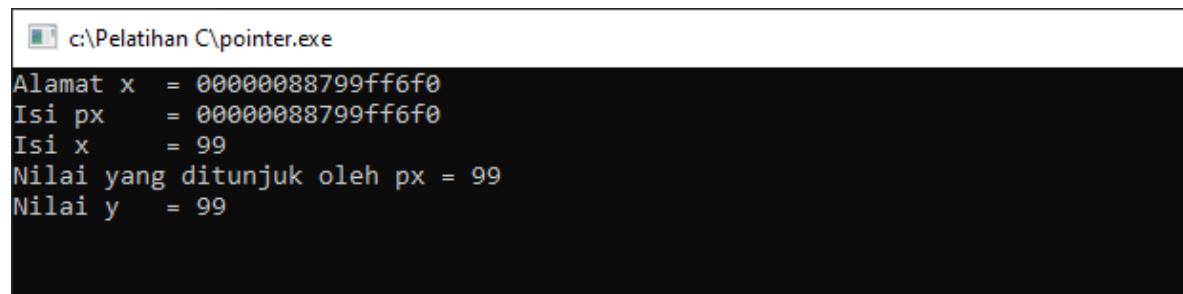
    // px diisi dengan alamat dari variabel x
    px = &x;

    // y diisi dengan nilai yg ditunjuk oleh px
    y = *px;

    printf("Alamat x = %p \n", &x);
    printf("Isi px = %p \n", px);
    printf("Isi x = %d \n", x);
    printf("Nilai yang ditunjuk oleh px = %d \n", *px);
    printf("Nilai y = %d \n", y);

    getch();
}
```

Output:



```
c:\Pelatihan C\pointer.exe
Alamat x = 00000088799ff6f0
Isi px = 00000088799ff6f0
Isi x = 99
Nilai yang ditunjuk oleh px = 99
Nilai y = 99
```

Variabel pointer harus bertipe data sama dengan variabel yang ditunjuk. Jika tidak maka akan mengembalikan hasil yang tidak diinginkan. Lebih jelasnya perhatikan kode program ini:

Contoh lainnya:

```
#include <stdio.h>

int main()
{
    // var pointer pu bertipe int
    int *pu;

    //var nu bertipe int
    int nu;

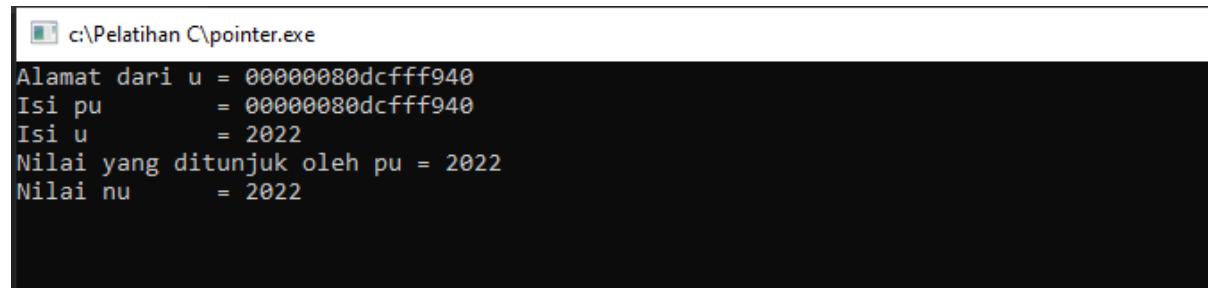
    // var u bertipe int
    int u = 2022;

    pu = &u;
    nu = *pu;

    printf("Alamat dari u = %p\n", &u);
    printf("Isi pu      = %p\n", pu);
    printf("Isi u      = %d\n", u);
    printf("Nilai yang ditunjuk oleh pu = %d\n", *pu);
    printf("Nilai nu     = %d\n", nu);

    getch();
}
```

Output:



```
c:\Pelatihan C\pointer.exe
Alamat dari u = 00000080dcfff940
Isi pu      = 00000080dcfff940
Isi u      = 2022
Nilai yang ditunjuk oleh pu = 2022
Nilai nu     = 2022
```

Perhatikan saat deklarasi variabel pointer tersebut, bahwa variabel pointer menunjukkan tipe data yang sama dengan variabel dan menghasilkan output yang sesuai. Tapi jika variabel dengan variabel pointer yang bertipe datanya berbeda maka akan menghasilkan hasil yang tidak sesuai. Lebih jelasnya perhatikan kode program ini:

```
#include <stdio.h>

int main()
{
    // var pointer pu bertipe int
    int *pu;

    // var nu bertipe int *
    int nu;

    // var u bertipe float
    float u = 2022;

    pu = &u;
    nu = *pu;

    printf("Alamat dari u = %p\n", &u);
    printf("Isi pu = %p\n", pu);
    printf("Isi u = %d\n", u);
    printf("Nilai yang ditunjuk oleh pu = %d\n", *pu);
    printf("Nilai nu = %d\n", nu);

    getch();
}
```

Output:

```
c:\Pelatihan C\pointer.exe
Alamat dari u = 000000cac17ff8f0
Isi pu = 000000cac17ff8f0
Isi u = 0
Nilai yang ditunjuk oleh pu = 1157414912
Nilai nu = 1157414912
```

Program ini menghasilkan output, namun outputnya tidak sesuai. Pada program akan menghasilkan problem yang seperti ini:

```
⚠ assignment to 'int **' from incompatible pointer type 'float *' [-Wincompatible-pointer-types] gcc [14, 12]
⚠ implicit declaration of function 'getch'; did you mean 'getc'? [-Wimplicit-function-declaration] gcc [23, 9]
```

Ini terjadi karena ada perbedaan deklarasi variabel yang berbeda dengan variabel pointer. Karena compiler tidak bisa mengubah tipe data float menjadi int. Untuk mengatasinya, kita perlu mengubah tipe data yang berbeda menjadi sama, seperti pada contoh program sebelumnya. Atau bisa juga mengubah tipe datanya menjadi float semua, tetapi untuk format specifier juga harus diubah dari "%d" menjadi "%f" dikarenakan variabel yang digunakan bertipe data float. Contoh:

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

int main(){
    // var pointer pu bertipe float */
    float *pu;

    // var nu bertipe float */
    float nu;

    // var u bertipe float */
    float u = 2022;
    pu = &u;
    nu = *pu;

    printf("Alamat dari u = %p\n", &u);
    printf("Isi pu = %p\n", pu);
    printf("Isi u = %f\n", u);
    printf("Nilai yang ditunjuk oleh pu = %f\n", *pu);
    printf("Nilai nu = %f\n", nu);

    getch();
}
```

Output:

```
c:\Pelatihan C\pointer.exe
Alamat dari u = 00000042603ffbc0
Isi pu = 00000042603ffbc0
Isi u = 2022.000000
Nilai yang ditunjuk oleh pu = 2022.000000
Nilai nu = 2022.000000
```

Jadi untuk menjalankan program di atas, variabel yang dideklarasikan harus sama.

Mencetak alamat dan mencetak nilai pointer

Contoh:

```
void main()
{
    int a = 27;
    int *b = &a;

    printf("Alamat dari a: %x\n", b);
    printf("Nilai dari a: %d\n", *b);
}
```

Output:

```
alamat dari a: 2d5ff684
nilai dari a: 27
```

Sebenarnya kita dapat bermain-main dengan mengulang-ulang simbol & dan * untuk mencetak nilai atau alamat dari variabel. Meskipun ini tidak berguna dalam program tapi ini bisa membantu kita dalam memahami cara kerja pointer.

Contoh:

```
int bilangan = 4;

printf("%d\n", bilangan); // mencetak nilai
printf("%d\n", &bilangan); // mencetak alamat
printf("%d\n", *&bilangan); // mencetak nilai
printf("%d\n", &*&bilangan); // mencetak alamat
printf("%d\n", * *&&bilangan); // mencetak nilai
printf("%d\n", &*&*&bilangan); // mencetak alamat
```

`&*&bilangan` artinya kita mengakses alamat dari variabel yang beralamat: alamat dari variabel yang beralamat: alamat dari variabel `bilangan`.

Output:

```
4
597686236
4
597686236
4
597686236
```

Perbedaan tanpa pointer dan menggunakan pointer

Ketika variabel pointer dicetak, maka yang tercetak adalah nilai yang berada di variabel yang ditunjuk. Jadi yang tercetak selalu nilai saat ini dari variabel yang ditunjuk.

Contoh:

```
void main()
{
    printf("tanpa pointer\n");
    int a = 2;
    int a2 = a;
    printf("nilai a2 sebelum a diubah: %d\n", a2);
    a = 3;
    printf("nilai a2 setelah a diubah: %d", a2);

    printf("\n\nmenggunakan pointer\n");
    int b = 2;
    int *b2 = &b;
    printf("nilai *b2 sebelum b diubah: %d\n", *b2);
    b = 3;
    printf("nilai *b2 setelah b diubah: %d", *b2);
}
```

Output:

```
tanpa pointer
nilai a2 sebelum a diubah: 2
nilai a2 setelah a diubah: 2

menggunakan pointer
nilai *b2 sebelum b diubah: 2
nilai *b2 setelah b diubah: 3_
```

Bab II

Linked List

Linked list?

- Linked list adalah salah satu bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dan dinamis.
- Linked list secara teori yaitu elemen yang berurutan yang dihubungkan dengan pointer. Paham tentang pointer? Jika sudah paham, mempelajari linked list gampang kok.
- Elemen terakhir menunjuk ke NULL.
- Elemen pada Single Linked List dapat bertambah atau berkurang (dinamis) selama program dijalankan.
- Dapat dibuat selama diperlukan (hingga memori sistem habis).
- Linked list lebih hemat memori dibandingkan array.

Sebelum memasuki lebih dalam tentang linked list, dalam bab sebelumnya kita harus memahami konsep dari struct pointer. Berikut penggunaan struct pointer:

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

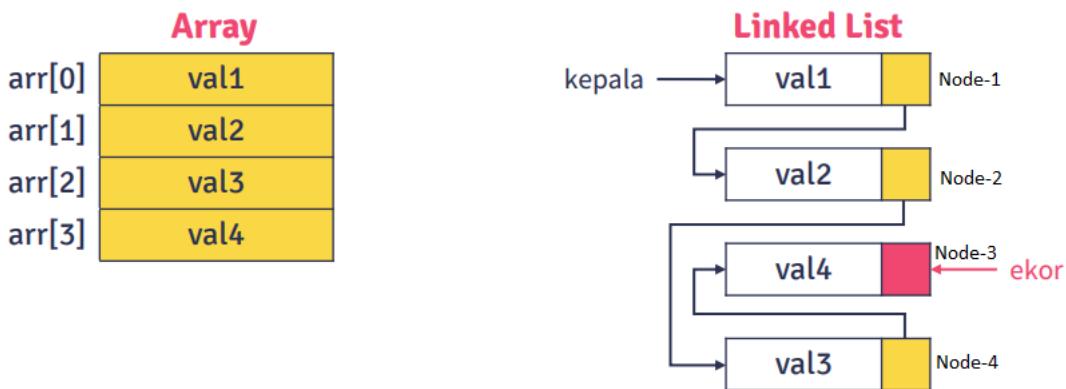
struct mahasiswa{
    int nim; //Deklarasi variabel bertipe int
    char *nama; //Deklarasi variabel bertipe String
};

int main(){
    struct mahasiswa *mhs; /*Disini var didefinisikan sebagai
variabel tipe struct dan pointer terlihat dari adanya (*)*/
    mhs=(struct mahasiswa*)malloc(sizeof(struct mahasiswa));
    /*malloc berfungsi untuk memesan tempat pada memori karena
pointer hanya menunjuk jadi perlu memori untuk menyimpannya
dan sizeof untuk menentukan memori yang dibutuhkan sama
dengan nama struct*/
    mhs->nim=11;
    mhs->nama="Dani Hidayat";
    /*Untuk mengakses variabel tersebut harus menggunakan
operator -> (panah)*/
    printf("Nim: %d\n",mhs->nim);
    printf("Nama: %s",mhs->nama);
    /*Untuk memanggilnya juga sama dengan menggunakan operator
panah*/
```

Array VS Linked List?

Array memiliki ruang atau aksesibilitas yang terbatas. Sedangkan linked list bisa mengalokasi memori secara dinamis.

- Perbandingan 1:

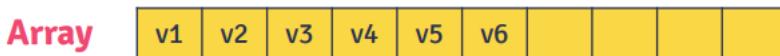


Array memiliki ruang atau aksesibilitas yang terbatas, value yang diisi tergantung berapa yang ingin kita deklarasikan, seperti contoh int arr[4], maka kita hanya bisa mengisi hingga 4 angka dan indeks yang kita isi selalu berurutan dari indeks ke-0 hingga ke-3.

Sedangkan linked list bisa, pada gambar terdapat 4 node yang saling terhubung, dari node-1 → node-2 → node-4 → node-3 → NULL / TAIL. Saling terhubung ini bisa dilakukan secara dinamis, menghubungkan serta menambahkan sesuai keinginan pengguna.

- Perbandingan 2:

Kita membuat array, int array[10] yang kita isi hanya 6 angka, 4 yang sisanya tetap disimpan di dalam memori, oleh karena itu penggunaan array disebut terbatas dan mubazir.



Linked list, menggunakan memori sesuai kebutuhan dan sesuai data yang ingin kita masukkan.

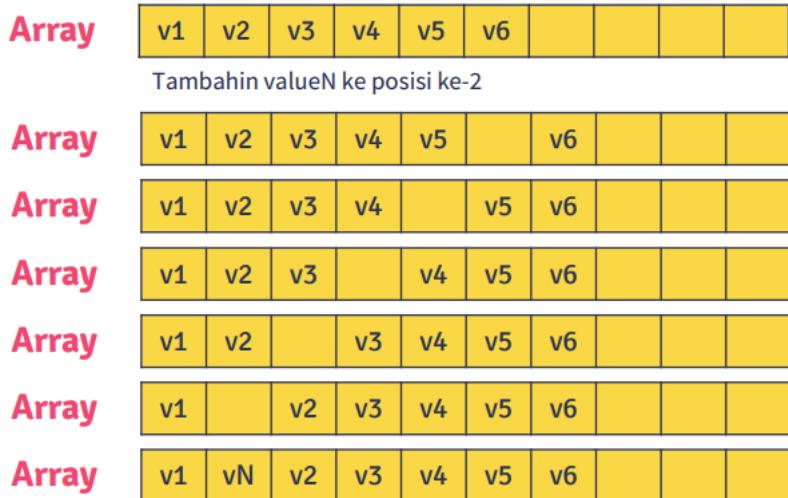


- Contoh masalah

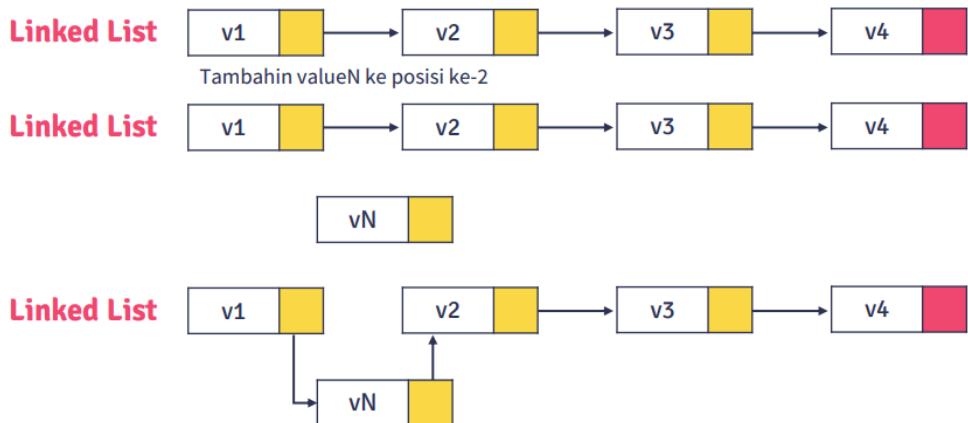
Terdapat sebuah array atau linked list, kita ingin menambahkan angka yang baru pada posisi ke-2, tentu saja angka yang berada pada posisi ke-2 ini tidak boleh ditimpa dan hanya boleh dipindahkan ke indeks/node setelahnya. Gimana cara array dan linked list menyelesaiakannya?

Markicob (Mari kita cobak)

Pada array:



Pada Linked list:



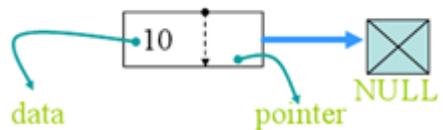
Sudah paham kan? Nah ini uraian singkat tentang perbedaan linked list dan array.

Array	Linked List
Array adalah kumpulan elemen dari tipe data yang sama.	Linked List adalah kumpulan berurutan dari elemen dengan tipe yang sama di mana setiap elemen terhubung ke elemen berikutnya menggunakan pointer.
Elemen array dapat diakses secara acak menggunakan indeks array.	Akses acak tidak dimungkinkan dalam daftar tertaut. Elemen harus diakses secara berurutan.
Elemen data disimpan di lokasi yang berdekatan dalam memori.	Elemen baru dapat disimpan di mana saja dan referensi dibuat untuk elemen baru menggunakan pointer.
Operasi penyisipan dan penghapusan lebih mahal karena lokasi memori berurutan dan tetap.	Operasi Penyisipan dan Penghapusan cepat dan mudah dalam linked list.
Memori dialokasikan selama waktu kompilasi (Alokasi memori statis).	Memori dialokasikan selama run-time (Alokasi memori dinamis).
Ukuran array harus ditentukan pada saat deklarasi/inisialisasi array.	Ukuran linked list tumbuh/menyusut saat dan ketika elemen baru dimasukkan/dihapus.

Bagian-Bagian Linked List

1. Linked list terdiri dari rantai node/ simpul di mana tiap node terdiri dari data, pointer yang menunjuk data node-nya sendiri dan pointer menunjuk pada node yang mengikutinya.
2. Linked list memiliki head/ kepala (node awal) dan tail/ekor (node akhir). Karena tail adalah node akhir maka pointer yang menunjuk selanjutnya sama dengan NULL (kosong) atau tidak menunjuk apapun.
3. Jika linked list hanya terdiri dari satu node maka dalam node tersebut yang menunjuk selanjutnya adalah sama dengan NULL.

Contoh:



Data dan pointer menjadi satu node seperti ilustrasi di atas. Jadi di dalam node terdapat penyimpanan data dan penunjuk alamat.



Pada node paling depan disebut **head** dan yang paling belakang disebut **tail**.
Linked list dapat di:

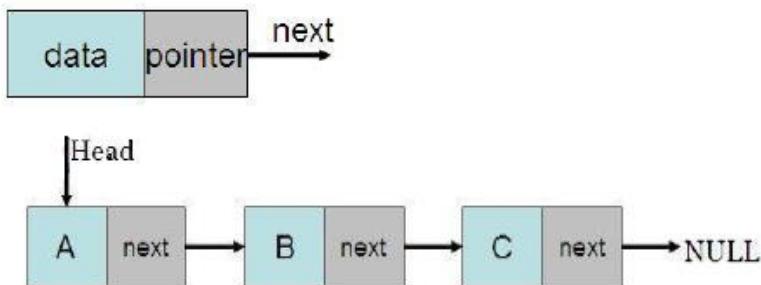
1. Update (Ubah)
2. Delete (Hapus)
3. Search (Cari)
4. Insert (Menambah/Masukan)
 - Depan
 - Belakang
 - Tengah
 - o Tengah banget
 - o By index
 - o By value

Jenis Linked List

- Single linked list
- Double linked list
- Circular linked list (dipelajari secara singkat)
- Multiple linked list (sekedar pengetahuan & tidak dipelajari dalam modul ini)

1. Single Linked List

Single Linked adalah apabila hanya ada satu pointer yang menghubungkan setiap node (satu arah "next"). Ilustrasi:

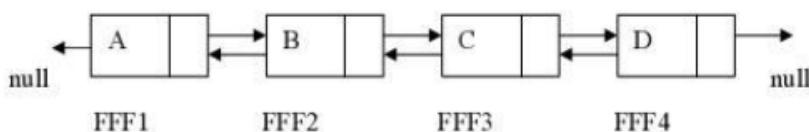


2. Double Linked List

Double Linked List adalah elemen-elemen yang dihubungkan dengan dua pointer dalam satu elemen dan list dapat melintas baik di depan atau belakang. Elemen double linked list terdiri dari 3 bagian:

1. Bagian data informasi
2. Pointer next yang menunjuk ke elemen berikutnya
3. Pointer prev yg menunjuk ke elemen sebelumnya

Ilustrasi:

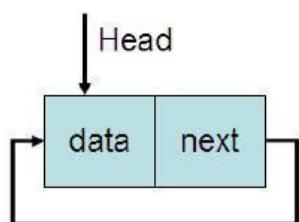


3. Circular Linked List

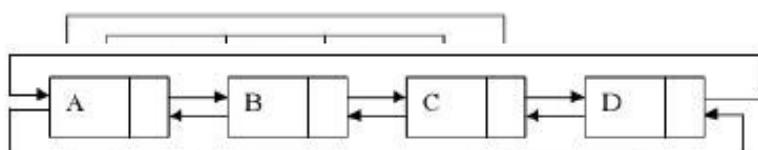
Bedanya dengan sebelumnya yaitu node terakhir akan menunjuk nilai NULL namun circular node terakhirnya menunjuk ke head lagi. Dibagi menjadi:

a. Single Linked List Circular

Hampir sama dengan single linked list sebelumnya, bahwa dibutuhkan sebuah kait untuk menghubungkan node-node data yang ada, di mana pada node terakhir atau tail yang semula menunjukkan NULL diganti dengan menunjuk ke kepala atau head. Ilustrasi:



b. Double Linked List Circular



Contoh:

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

struct node{
    int angka;
    struct node *next;
};

struct node *x,*y,*z; /*Pendeklarasian variabel pointer bertipe
node*/



void tampil(){
    struct node *bantu;
    bantu=x;
    while(bantu!=NULL){
        printf("|%d|-->",bantu->angka);
        bantu=bantu->next;
    }
    printf("NULL\n");
}

void awal(int a, int b, int c){
    x=(struct node*)malloc(sizeof(struct node));
    y=(struct node*)malloc(sizeof(struct node));
    z=(struct node*)malloc(sizeof(struct node));
    x->angka = a;
    y->angka = b;
    z->angka = c;
    x->next = y;
    y->next = z;
    z->next = NULL;
    tampil();
}

int main(){
    int a,b,c;
    printf("Masukkan Nilai 1:");scanf("%d",&a);
    printf("Masukkan Nilai 2:");scanf("%d",&b);
    printf("Masukkan Nilai 3:");scanf("%d",&c);
    awal(a,b,c);
    return 0;
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
Masukkan Nilai 1 :12
Masukkan Nilai 2 :10
Masukkan Nilai 3 :2002
|12|-->|10|-->|2002|-->NULL
```

Dari penjelasan di atas terdapat pendeklarasian struct bernama **node** berisi **int angka** dan **node pointer next**. Jadi node tersebut memiliki 2 tipe data yaitu **int** dan **node** (struct itu sendiri). Variabel **angka** tersebut digunakan untuk menampung nilai bertipe **int**. Lalu variabel pointer **next** untuk menunjuk node berikutnya, itulah kenapa **next** bertipe variabel pointer. Lalu kenapa bertipe data **node**? Karena yang ditunjuk oleh **next** tersebut menunjuk ke alamat node selanjutnya.

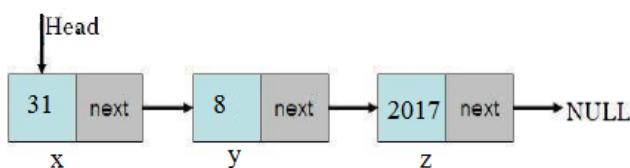
Dari kodingan tersebut terdapat fungsi **malloc** dan **size**. **Malloc** berfungsi menyamakan antara variabel pointer dengan structnya, atau bisa juga untuk menyiapkan tempat pada memori untuk menampung suatu nilai di variabel pointer. Ini dikarenakan pointer tidak bisa menyimpan nilai, hanya bisa menunjuk suatu alamat. **Size** berfungsi untuk menentukan ukuran memori yang dibutuhkan oleh variabel tersebut.

```
x=(node*)malloc(sizeof(node));
y=(node*)malloc(sizeof(node));
z=(node*)malloc(sizeof(node));

x->angka = a;
y->angka = b;
z->angka = c;
x->next = y;
y->next = z;
z->next = NULL;
```

Dari kode di atas variabel **x,y,z** tersebut di **malloc**. Lalu diikuti oleh **x->angka = a** jadi x menampung nilai a melalui angka. Dan begitu juga dengan y dan z. Lalu **x -> next = y** berarti variabel x menunjuk variabel y melalui next. Begitu juga dengan y dan z. Namun pada variabel terakhir menujuk kepada **NULL**, karena tidak ada lagi veriabel yang ditunjuk (dihubungkan).

Ilustrasi:



Operasi-operasi dalam Linked List

Sebelumnya untuk mendukung kode-kode yang di bawah ini dibutuhkan struct berikut kodenya:

```
struct node{
    int angka;
    struct node *next;
}*head,*baru,*bantu,*tamp;
```

Dan juga pendeklarasian variabel pointer head, baru, bantu, tamp yang bertipe struct node.

1. Insert Node (Tambah)

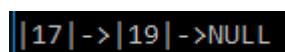
Dalam menambah node pada linked ada tiga macam yaitu menambah dari depan, tengah, dan belakang. Disini hanya akan diberi contoh depan dan belakang.

a. Insert Depan (tambah depan)

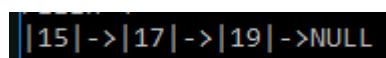
Jadi jika kita menginputkan data ke dalam linked list akan tercetak paling depan. Berikut contoh kode:

```
void tambahDepan(int data){
    baru=(struct node*)malloc(sizeof( struct node));
    baru->angka=data;
    if(head==NULL){
        head=baru;
        head->next=NULL;
    }else{
        baru->next=head;
        head=baru;
    }
}
```

Sebelum:



Sesudah:



Penjelasan:

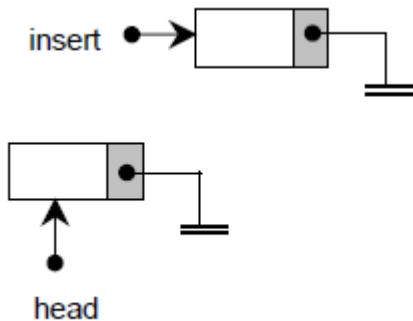
Terdapat fungsi tambahDepan yang mempunyai parameter **data** bertipe data int. Lalu terdapat variabel baru yang di malloc yang berfungsi untuk dapat variabel pointer untuk menyimpan suatu nilai. Lalu terdapat **baru->angka=data** yang dimaksud menginisialisasi variabel baru yang disimpan melalui angka. Selanjutnya ada kondisi di mana variabel head itu NULL atau bukan, jika head masih NULL maka masuk ke dalam kondisi tersebut, dan melakukan statement bahwa head itu sama dengan baru. Lalu **head->next** untuk menunjuk

ke node berikutnya karena sebelumnya head masih kosong maka head langsung menunjuk ke NULL. Kalau head sudah memiliki isian maka masuk **else**, dan melakukan statement bahwa baru tersebut menunjuk ke head terlebih dahulu dengan **baru->next=head**. Selanjutnya ubah baru yang di depan menjadi head. Tambahan head harus selalu paling depan.

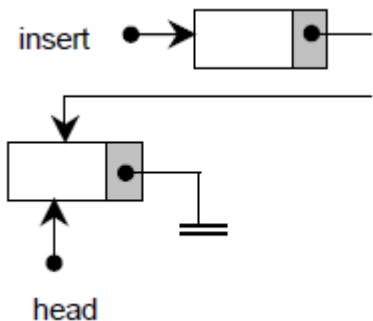
Ilustrasi:

Catatan nyatakan bahwa insert di sini adalah baru

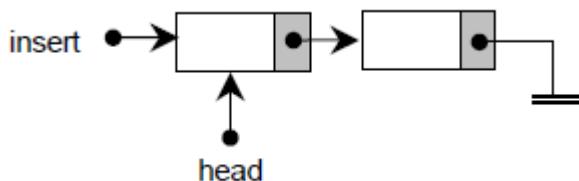
Kondisi awal:



`insert->next=head;`



`head=insert;`



b. Insert Belakang (tambah belakang)

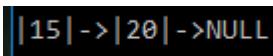
Jika kita menginputkan data baru ke dalam linked list akan tercetak paling belakang. Ini diperlukan variabel pointer baru sebagai pembatu. Sehingga head tetap diposisi depan. Contoh kode:

```

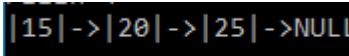
void tambahBelakang(int data){
    baru = (struct node*)malloc(sizeof(struct node));
    baru->angka = data;
    if(head == NULL){
        head = baru;
        head->next = NULL;
    }else{
        bantu = head;
        while(bantu->next != NULL){
            bantu = bantu->next;
        }
        bantu->next = baru;
        baru->next = NULL;
    }
}

```

Sebelum:



Sesudah:



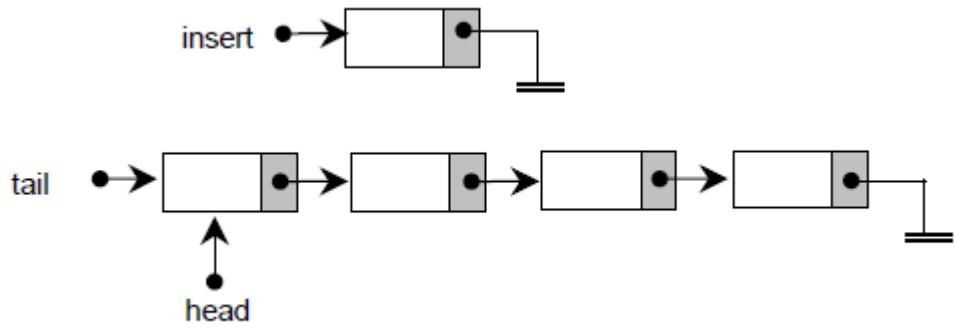
Penjelasan:

Terdapat fungsi yang bernama tambahBelakang yang mempunyai parameter data bertipe int. Mempunyai statement awal yang sama dengan tambahDepan maka langsung saja ke **else** buat variabel bantu itu sama dengan head, lalu terdapat perulangan while untuk mencari kondisi node paling belakang. Terdapat kondisi while (**bantu->next!=NULL**) itu berarti perulangan tidak akan berhenti sebelum menemukan bantu->next itu NULL, dan di dalam while terdapat statement **bantu=bantu->next** berguna untuk mengubah posisi letak bantu dari depan menuju node paling terakhir tanpa mengubah posisi letak head. Disaat sudah ditemukan maka perulangan akan berhenti dan menjalankan stement berikutnya. Di node terakhir bantu menunjuk ke baru dengan **bantu->next=baru**, lalu letak bantu menjadi sama dengan letak dari baru. Setelah itu bantu menunjuk ke NULL.

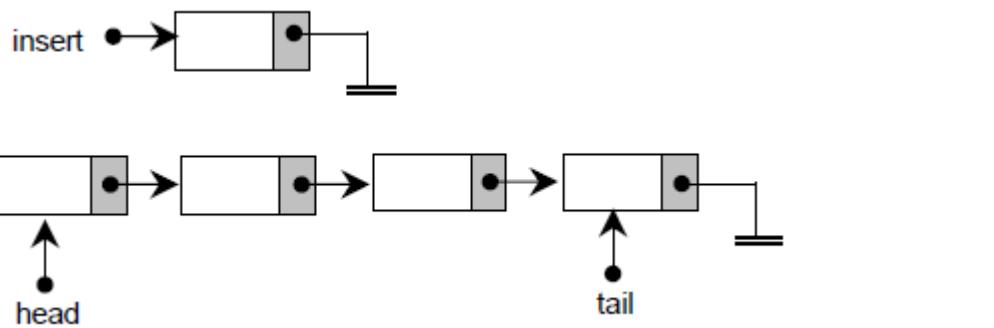
Ilustrasi:

Nyatakan bahwa insert adalah baru dan tail di sini adalah bantu

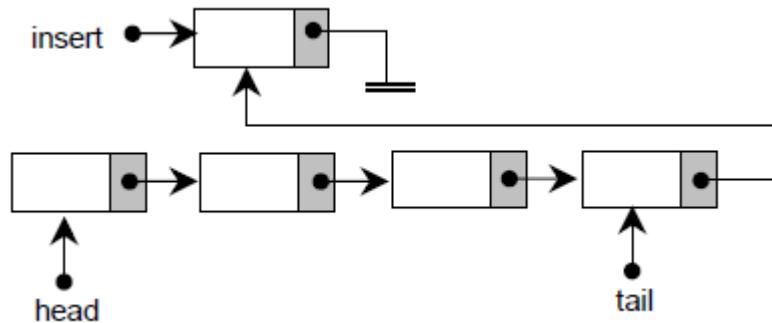
Kondisi awal:



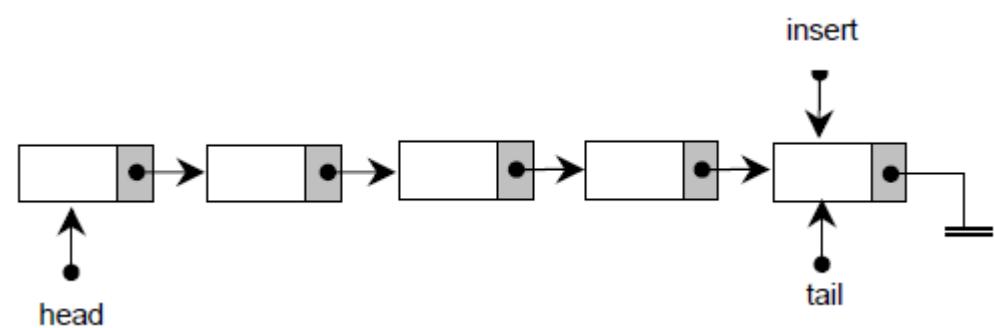
```
while(bantu->next!=NULL){
    bantu=bantu->next;
}
```



```
bantu->next=baru;
```



```
bantu=baru;
bantu->next=NULL;
```



2. Delete Node (Menghapus)

Operasi hapus node juga ada 3 depan, belakang, dan tengah. Untuk menghapus suatu node maka diperlukan variabel pointer baru sebagai pembantu.

a. Hapus Depan

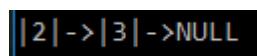
Jika kita menghapus depan maka yang akan terhapus adalah data paling depan. Contoh kode:

```
void hapusDepan(){
    if(head == NULL){
        printf("Data Kosong\n");
    }else{
        tamp = head;
        head = head->next;
        free(tamp);
    }
}
```

Sebelum:



Sesudah:



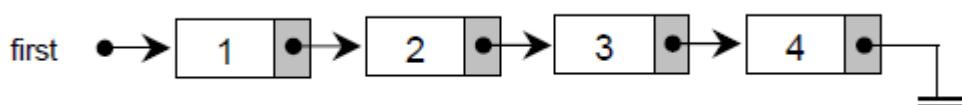
Penjelasan:

Terdapat suatu fungsi bernama hapusDepan tidak mempunyai suatu parameter. Langsung terdapat kondisi yang menyatakan bahwa head itu NULL atau bukan kalau NULL maka head masih kosong dan memberikan keterangan "Data Belum Ada". Ada **else** yang menyatakan temp itu sama dengan head, lalu head megubah node-nya ke node selanjutnya. Sehingga head tidak lagi menunjuk node sebelumnya. Lalu ada delete untuk mensosongkan memori, karena menggeser head sebenarnya node masih terhubung hanya tidak kebaca karena yang linked list dimulai dari head. Karenanya data harus ditunjuk oleh tamp untuk dikosongkan lagi memorinya.

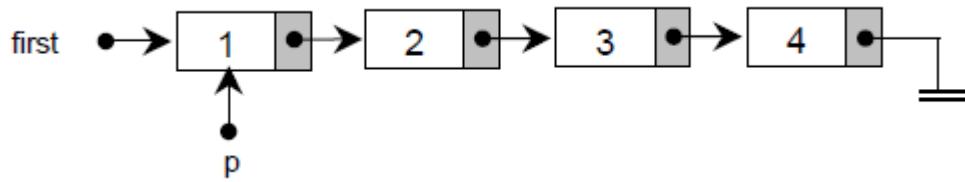
Ilustrasi:

Nyatakan bahwa first adalah head dan tamp adalah p

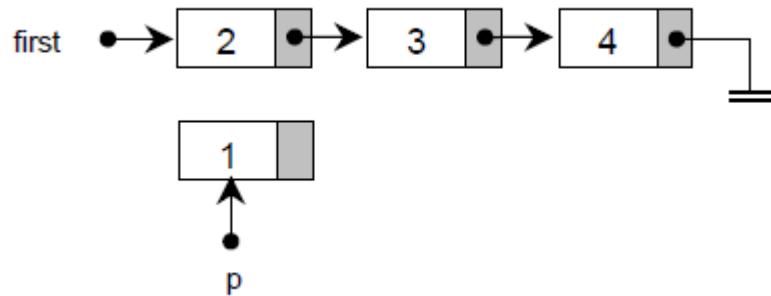
Kondisi awal:



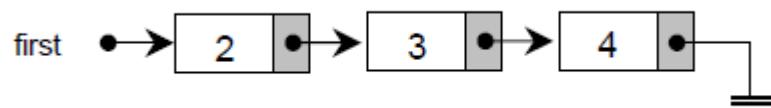
p=head;



head=head->next;



free(tamp);



b. Hapus Belakang

Data yang terhapus adalah data yang letaknya paling belakang. Contoh kode:

```
void hapusBelakang(){
    if(head == NULL){
        printf("Data Kosong\n");
    }else if(head->next == NULL){
        free(head);
        head = NULL;
    }else{
        bantu = head;
        while(bantu->next->next != NULL){
            bantu = bantu->next;
        }
        free(bantu->next);
        bantu->next = NULL;
    }
}
```

Sebelum:

```
| 2 | ->| 9 | ->| 17 | ->NULL
```

Sesudah:

```
| 2 | ->| 9 | ->NULL
```

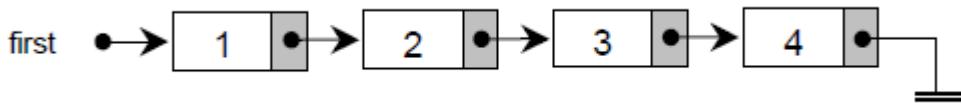
Penjelasan:

Terdapat fungsi hapusBelakang tanpa parameter, karena kondisi awal seperti hapusDepan langsung kondisi else. Nyatakan bantu itu adalah head, lalu terdapat kondisi lagi di mana hanya ada satu data yaitu bantu->next itu NULL, maka langsung delete dan head menjadi NULL. Selanjutnya else terdapat while di mana kondisinya yaitu (bantu->next->next!=NULL) berarti untuk mencari 2 node terakhir. Jika sudah ketemu node paling belakang yaitu bantu->next->next dihapus, lalu bantu->next menunjuk ke NULL.

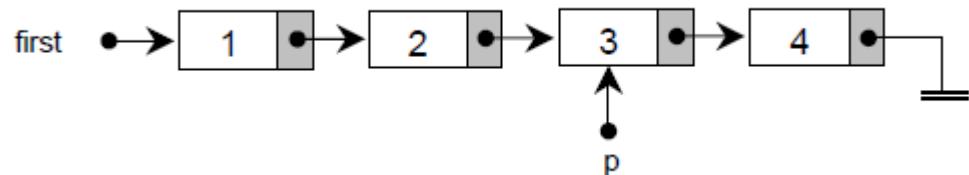
Ilustrasi:

Nyatakan bahwa first adalah head dan bantu adalah p

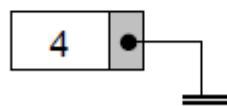
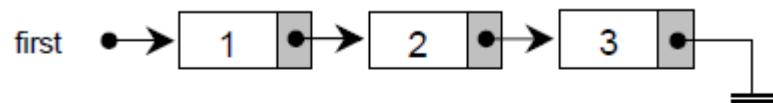
Kondisi awal:



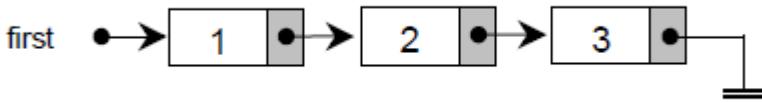
```
bantu=head;
while(bantu->next->next!=NULL){
    bantu=bantu->next;
}
```



```
free(bantu->next);
```



```
bantu->next=NULL;
```



3. Mencetak Linked List

Kita tidak bisa mengingat atau mengetahui isi linked list jika kita tidak mencetaknya. Contoh kode:

```
void cetak(){
    bantu=head;
    while(bantu!=NULL){
        printf("|%d| ->", bantu->angka);
        bantu=bantu->next;
    }
    printf("NULL\n");
}
```

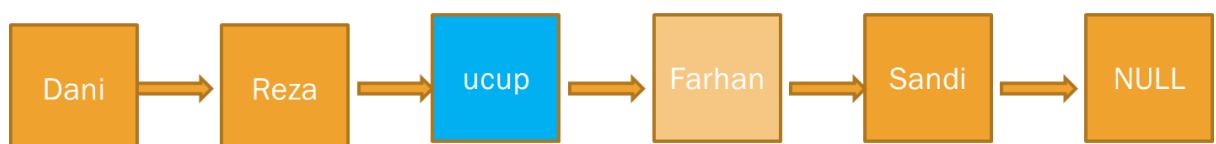
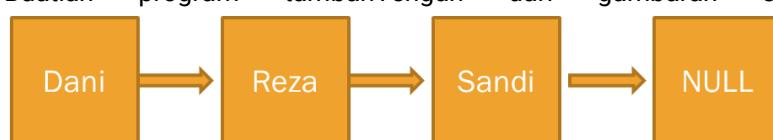
Contoh, jika kita menambah depan data secara berturut-turut seperti **tambahDepan(22)** lalu **tambahDepan(5)** selanjutnya **tambahDepan(3)**.

Maka akan menghasilkan output:

```
|3| -> |5| -> |22| ->NULL ]
```

Latihan:

- Buatlah program tambahDepan & hapusDepan yang hanya menerima inputan ganjil
- Buatlah program tambahBelakang & hapusBelakang yang hanya menerima inputan tahun kabisat
- Buatlah program tambahDepan, tambahBelakang, hapusDepan & hapusBelakang yang hanya menerima inputan bilangan prima
- Buatlah program tambahTengah dari gambaran seperti di bawah ini:



4. Update

Dalam update terbagi atas dua yaitu:

- Update By Index (int angka, int index)

Dalam update By Index, mengubah angka yang dimasukkan sesuai dengan index yang diinginkan. Semisalnya updateByIndex(10,1) ini berarti bahwa mengubah isi dari index ke-1 ditukar menjadi angka 10.

```
void main(){
    tambahDepan(1);
    tambahDepan(2);
    tambahDepan(3);
    tambahDepan(4);
    cetak();
    updateByIndex(10,1);
    cetak();
}
```

```
C:\WINDOWS\system32\cmd.exe
|4|->|3|->|2|->|1|->NULL
|4|->|10|->|2|->|1|->NULL
```

- Update By Value (int angka, int value)

Dalam update By Value, mengubah angka yang dimasukkan sesuai value yang diinginkan. Semisal kita ingin updateByValue(10,3) berarti mengubah value yang angkanya 3 menjadi angka 10, ini berlaku hanya untuk angka yang ditemukannya pertama kali, jadi jika ada dua buah angka 3 maka angka 3 yang pertama yang akan di-update.

```
void main(){
    tambahBelakang(1);
    tambahBelakang(1);
    tambahBelakang(2);
    tambahBelakang(2);
    tambahBelakang(3);
    tambahBelakang(3);
    tambahBelakang(3);
    cetak();
    updateByValue(10,3);
    cetak();
}
```

```
C:\WINDOWS\system32\cmd.exe
|1|->|1|->|2|->|2|->|3|->|3|->NULL
|1|->|1|->|2|->|2|->|10|->|3|->NULL
```

Latihan:

Buatlah program

- updateByIndex(int angka, int index)
- updateByValue(int angka, int value)

Latihan Final:

Buatlah program single linked list dan selesaikan semua program di bawah ini.

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

// Prototype
void tambah();
void hapus();
void update();
void cek();

void tambahDepan();
void tambahBelakang();
void tambahByIndex();
void tambahByValue();
void tambahTengah();

void hapusDepan();
void hapusBelakang();
void hapusByIndex();
void hapusByValue();
void hapusTengah();

void updateByIndex();
void updateByValue();

void tampil();
void cek_max();
void cek_min();
void cek_average();
void cek_sum();

// Program Linked List

struct node{
    int angka;
    struct node *next;
}*head,*baru,*bantu,*bantu2,*tam;
```

```
void ngulang(){
    int pilih;
    printf("=====\\n");
    printf("===== LINGKED =====\\n");
    printf("===== LIST =====\\n");
    printf("=====<<_____>> =====\\n");
    printf("=====\\n");
    printf("= 1. TAMBAH DATA =\\n");
    printf("= 2. HAPUS DATA =\\n");
    printf("= 3. UPDATE DATA =\\n");
    printf("= 4. CEK DATA =\\n");
    printf("= 5. KELUAR =\\n");
    printf("= =\\n");
    printf("PILIH: ");scanf("%d",&pilih);
    printf("=====\\n");
    if(pilih==1)
        tambah();
    else if(pilih==2)
        hapus();
    else if(pilih==3)
        update();
    else if(pilih==4)
        cek();
    else if(pilih==5)
        exit(0);
    else
        ngulang();
}
```

```
void tambah(){
    int pilih;
    printf("=====\\n");
    printf("= 1. TAMBAH DEPAN      =\\n");
    printf("= 2. TAMBAH BELAKANG   =\\n");
    printf("= 3. TAMBAH TENGAH BY INDEX =\\n");
    printf("= 4. TAMBAH TENGAH BY VALUE  =\\n");
    printf("= 5. TAMBAH TENGAH BANGET   =\\n");
    printf("= 6. KEMBALI           =\\n");
    printf("=\\n");
    printf("      PILIH: ");scanf("%d",&pilih);
    printf("=====\\n");
    if(pilih==1)
        tambahDepan();
    else if(pilih==2)
        tambahBelakang();
    else if(pilih==3)
        tambahByIndex();
    else if(pilih==4)
        tambahByValue();
    else if(pilih==5)
        tambahTengah();
    else if(pilih==6)
        ngulang();
    else
        tambah();
    tampil();
    tambah();
}
```

```

void hapus(){
    int pilih;
    printf("=====\\n");
    printf("= 1. HAPUS DEPAN      =\\n");
    printf("= 2. HAPUS BELAKANG   =\\n");
    printf("= 3. HAPUS TENGAH BY INDEX =\\n");
    printf("= 4. HAPUS TENGAH BY VALUE  =\\n");
    printf("= 5. HAPUS TENGAH BANGET =\\n");
    printf("= 6. KEMBALI          =\\n");
    printf("=\\n");
    printf("      PILIH: ");scanf("%d",&pilih);
    printf("=====\\n");
    if(pilih==1)
        hapusDepan();
    else if(pilih==2)
        hapusBelakang();
    else if(pilih==3)
        hapusByIndex();
    else if(pilih==4)
        hapusByValue();
    else if(pilih==5)
        hapusTengah();
    else if(pilih==6)
        ngulang();
    else
        hapus();
    tampil();
    hapus();
}

```

```

void update(){
    int pilih;
    printf("=====\\n");
    printf("= 1. UPDATE BY INDEX      =\\n");
    printf("= 2. UPDATE BY VALUE     =\\n");
    printf("= 3. KEMBALI             =\\n");
    printf("=                         =\\n");
    printf("      PILIH: ");scanf("%d",&pilih);
    printf("=====\\n");
    if(pilih==1)
        updateByIndex();
    else if(pilih==2)
        updateByValue();
    else if(pilih==3)
        ngulang();
    else
        update();
    tampil();
    update();
}

void tampil(){
    printf("=====\\n");
    tam=head;
    while(tam!=NULL){
        printf(" |%d| -> ",tam->angka);
        tam=tam->next;
    }
    printf("NULL\\n");
    printf("=====\\n");
}

void main(){
    ngulang();
}

```

```
void cek(){
    int pilih;
    printf("=====\\n");
    printf("= 1. CEK DATA LINKED LIST    =\\n");
    printf("= 2. ANGKA TERBESAR        =\\n");
    printf("= 3. ANGKA TERKECIL        =\\n");
    printf("= 4. RATA-RATA             =\\n");
    printf("= 5. JUMLAH                 =\\n");
    printf("= 6. KEMBALI                 =\\n");
    printf("=\\n");
    printf("      PILIH: ");scanf("%d",&pilih);
    printf("=====\\n");
    if(pilih==1)
        tampil();
    else if(pilih==2)
        cek_max();
    else if(pilih==3)
        cek_min();
    else if(pilih==4)
        cek_average();
    else if(pilih==5)
        cek_sum();
    else if(pilih==6)
        ngulang();
    cek();
}
```

```

// Linked List Code
// Tambah

void tambahDepan(){
    int angka;
    printf("Masukan angka yang ingin ditambah: ");scanf("%d",&angka);

    baru=(struct node*)malloc(sizeof(struct node));
    baru->angka=angka;
    baru->next=head;
    head=baru;
}

void tambahBelakang(){
    int angka;
    printf("Masukan angka yang ingin ditambah: ");scanf("%d",&angka);

    baru = (struct node*)malloc(sizeof(struct node));
    baru->angka = angka;
    if(head == NULL){
        head = baru;
        head->next = NULL;
    }else{
        bantu = head;
        while(bantu->next != NULL){
            bantu = bantu->next;
        }
        bantu->next = baru;
        baru->next = NULL;
    }
}

void tambahByIndex(){
    // . .
}

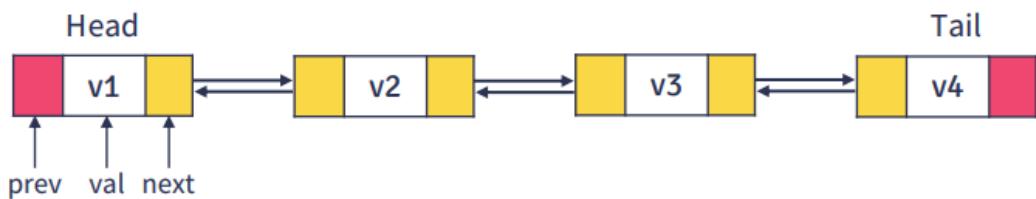
void tambahByValue(){
    // . .
}

// Dan lain-lain

```

Double Linked List?

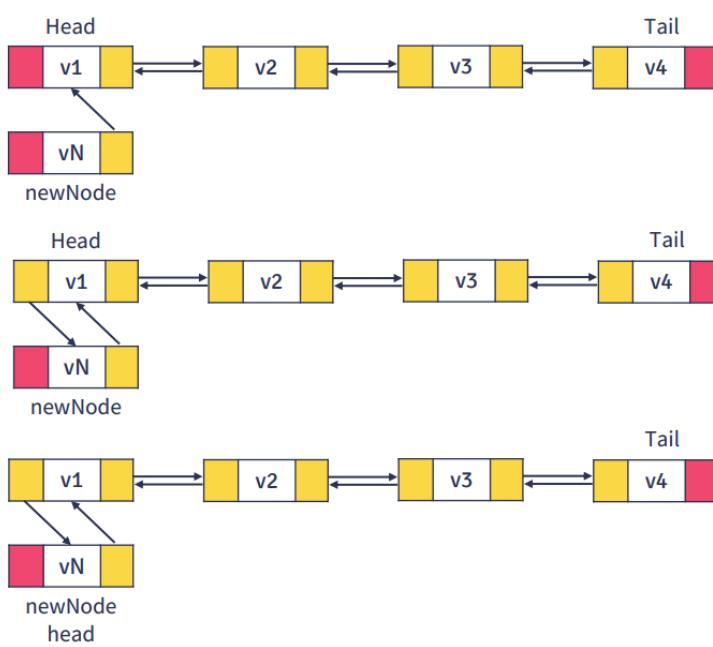
- Double Linked List merupakan suatu linked list yang memiliki dua variabel pointer. Di mana pointer tersebut menunjuk ke node sebelum dan selanjutnya (prev & next).
- Double Linked List terdiri dari sejumlah elemen (node) di mana setiap node memiliki penunjuk prev (menunjuk node sebelumnya) dan next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



Contoh penggunaan struct dalam double linked list:

```
struct node{  
    int angka;  
    struct node *next;  
    struct node *prev;  
}*head,*tail,*baru,*tam,*tamp;
```

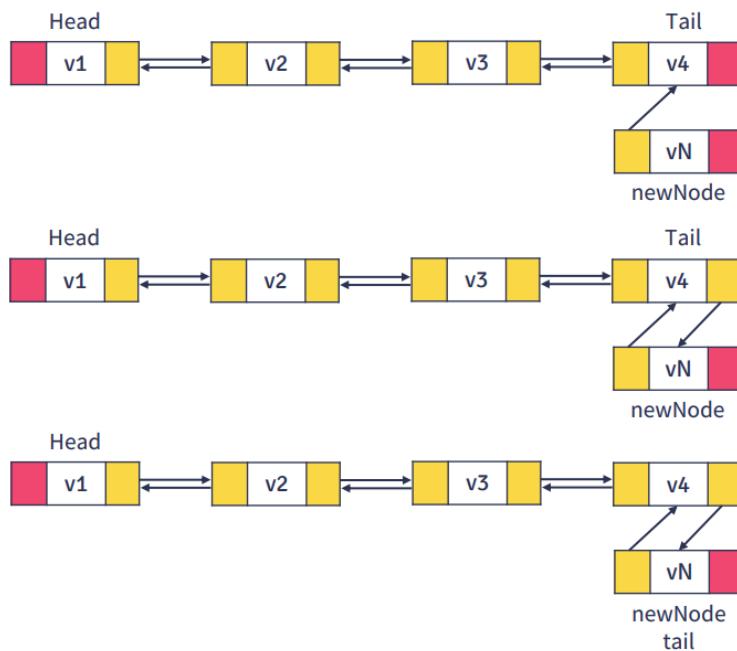
1. Tambah depan



Source code:

```
void tambahDepan(int a){  
    baru=(struct node*)malloc(sizeof(struct node));  
    baru->angka=a;  
    if(head==NULL){  
        tail=head;  
        head=tail=baru;  
    }else{  
        baru->next=head;  
        head->prev=baru;  
        head=baru;  
    }head->prev=NULL;  
    tail->next=NULL;  
}
```

2. Tambah Belakang



Source code:

```
void tambahBelakang(int a){
    baru=(struct node*)malloc(sizeof(struct node));
    baru->angka=a;
    if(head==NULL){
        head=tail=baru;
    }else{
        tail->next=baru;
        baru->prev=tail;
        tail=baru;
    }head->prev=NULL;
    tail->next=NULL;
}
```

3. Mencetak isi dari double linked list

```
void tampilD(){ // Tampil dari depan
    tam=head;
    while(tam!=NULL){
        printf("%d->",tam->angka);
        tam=tam->next;
    }printf("NULL\n");
}
void tampilB(){ // Tampil dari belakang
    tam=tail;
    while(tam!=NULL){
        printf("%d->",tam->angka);
        tam=tam->prev;
    }printf("NULL\n");
}
```

Output:

```
void main(){
    tambahDepan(1);
    tambahDepan(2);
    tambahDepan(3);
    tambahDepan(4);
    tambahDepan(5);
    tampilD();
    tampilB();
}
```

C:\WINDOWS\system32\cmd.exe
|5|->|4|->|3|->|2|->|1|->NULL
|1|->|2|->|3|->|4|->|5|->NULL

Latihan:

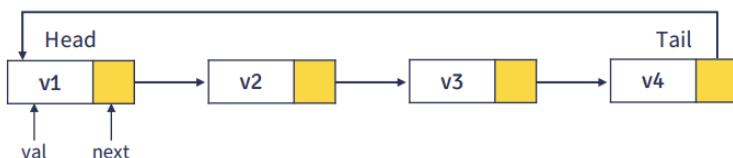
- 1) Buatlah program double linked list tentang hapusDepan() dan hapusBelakang()
- 2) Buatlah program double linked list tentang tambahByValue() dan tambahByIndex()

Circular Linked List?

Circular Linked List adalah sekumpulan node atau simpul yang tidak terdapat nilai NULL pada satupun nodenya.

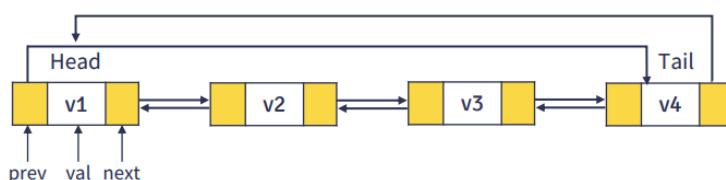
1. Circular Single Linked List

Bentuk node pada Circular Single Linked List sama dengan Single Linked List yang mempunyai data dan pointer next. Yang membedakan adalah jika pada Single Linked List node terakhir menunjuk ke NULL, tapi di Circular Single Linked List node terakhir menunjuk ke node pertama.



2. Circular Double Linked List

Struktur node sama seperti Double Linked List yang mana terdapat data, pointer prev dan pointer next. Karena ini maka pointer next pada tail menunjuk ke head dan pointer prev pada head menunjuk ke tail.



STACK

Stack bisa diartikan sebagai suatu kumpulan data yang seolah-olah ada data yang diletakkan di atas data yang lain. Contoh penerapan stack dalam kehidupan sehari-hari yaitu tumpukan piring, di mana piring yang terakhir masuk atau ditumpuk akan berada di posisi paling atas, sementara ketika kita akan mengeluarkan atau mengambil piring, yang pertama diambil pasti piring yang paling atas yaitu piring yang terakhir masuk atau ditumpuk. Contoh lain yaitu tumpukan buku matematika, fisika, biologi, dan kimia. Buku matematika kamu masukkan terlebih dahulu, lalu buku selanjutnya fisika, biologi, dan kimia. Nah, untuk mendapatkan buku fisika, kamu harus mengeluarkan buku kimia dan biologi terlebih dahulu, karena kedua buku tersebut tergolong buku yang terakhir masuk.

Berdasarkan uraian di atas stack memiliki prinsip LIFO (Last In First Out), yaitu yang terakhir masuk adalah yang pertama keluar. Kita bisa menambahkan (menyisipkan) dan mengambil (menghapus) data melalui ujung yang sama yang disebut sebagai ujung atas stack (top of stack).

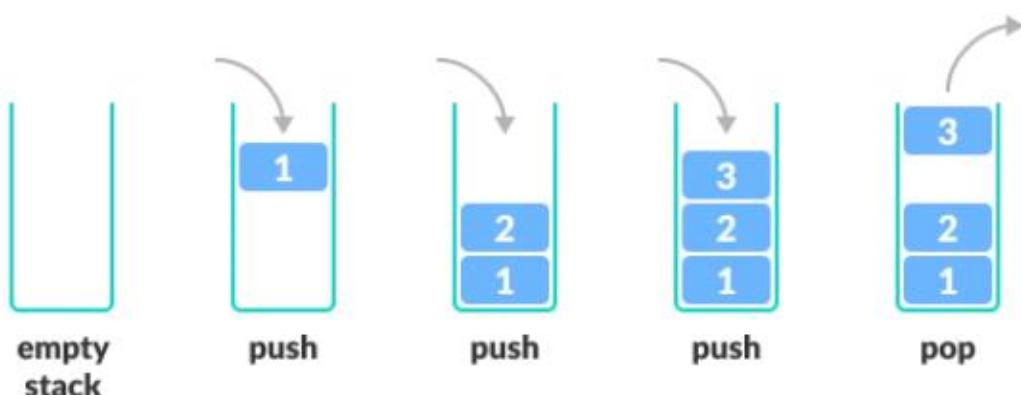
Bentuk penyajian stack bisa menggunakan tipe data array, tetapi sebenarnya penyajian stack menggunakan array adalah kurang tepat karena banyaknya elemen dalam array adalah statis, sedangkan dalam stack banyaknya elemen sangat bervariasi atau dinamis. Meskipun demikian, array bisa digunakan untuk penyajian stack, tetapi dengan anggapan bahwa banyaknya elemen maksimal dari suatu stack tidak melebihi batas maksimum banyaknya elemen array.

Pada suatu saat, ukuran stack akan sama dengan ukuran array. Bila diteruskan menambah data, maka akan terjadi overflow.

Operasi-operasi yang biasanya terdapat pada Stack yaitu:

1. Push : digunakan untuk menambah item pada stack atau tumpukan paling atas
2. Pop : digunakan untuk mengambil item pada stack atau tumpukan paling atas
3. Clear : digunakan untuk mengosongkan stack
4. IsEmpty : fungsi yang digunakan untuk memeriksa apakah stack kosong
5. IsFull : fungsi yang digunakan untuk memeriksa apakah stack penuh

Secara sederhana, stack/tumpukan dapat digambarkan seperti berikut:



Implementasi stack dengan Array:

The screenshot shows a code editor interface with two tabs at the top: 'stack.c' and 'queue.c'. The left sidebar contains icons for file operations like Open, Save, Find, Copy, Paste, and Help. The right pane displays the content of the 'stack.c' file.

```
File Edit Selection View Go Run Terminal Help
C stack.c 1 X C queue.c
C stack.c > ...
1 #include<stdio.h>
2 #include<conio.h>
3
4 int stack[10];
5 int index = -1;
6
7 int isEmpty(){
8     if(index == -1){
9         return 1;
10    }else{
11        return 0;
12    }
13 }
14
15 int isFull(){
16     if(index == 9){
17         return 1;
18     }else{
19         return 0;
20     }
21 }
22
23 void push(int data){
24     if(isFull() == 1){
25         printf("Stack sudah penuh \n");
26     }else{
27         index++;
28         stack[index] = data;
29         printf("Data berhasil disimpan \n");
30     }
31 }
32
33 void pop(){
34     if(isEmpty() == 1){
35         printf("Stack kosong \n");
36     }else{
37         printf("Data terambil: %d\n", stack[index]);
38         index--;
39     }
40 }
41
42 void tampil(){
43     for(int a=index; a>=0; a--){
44         printf("%d \n", stack[a]);
45     }
46 }
47
48 void main(){
49     push(0);
50     push(1);
51     push(2);
52     push(3);
53     pop();
54     push(4);
55     tampil();
56     getch();
57 }
```

Penjelasan:

```
int stack[10];
int index = -1;
```

Pendeklarasian array yang bernama stack dengan panjang 10. Ini digunakan untuk menampung data stack yang berarti data maksimum stack yang masuk adalah 10 data.

Pendeklarasian dan inisialisasi variable index yang bertipe integer dengan nilai -1. Ini digunakan untuk mengosongkan array atau sebagai tanda bahwa array masih belum diisi.

```
int isEmpty(){
    if(index == -1){
        return 1;
    }else{
        return 0;
    }
}
```

Fungsi isEmpty digunakan untuk mengecek apakah array dalam keadaan kosong atau tidak yang artinya nilai variable index ==-1, maka dia akan mereturn nilai 1 dan jika array tidak kosong maka fungsi ini akan mereturn nilai 0.

```
int isFull(){
    if(index == 9){
        return 1;
    }else{
        return 0;
    }
}
```

Fungsi ini digunakan untuk mengecek array apakah sudah terisi penuh. Jika index==9 yang artinya index array sudah terisi penuh karena telah mencapai index ke 9 maka fungsi ini akan mereturn 1 sebaliknya fungsi ini akan mereturn 0.

```
void push(int data){
    if(isFull() == 1){
        printf("Stack sudah penuh \n");
    }else{
        index++;
        stack[index] = data;
        printf("Data berhasil disimpan \n");
    }
}
```

Fungsi push ini adalah untuk menambah data dalam stack. Pertama fungsi ini akan mengecek menggunakan fungsi isFull apakah array penuh atau tidak. Jika sudah penuh berarti kita tidak bisa menambah data. Jika array belum terisi penuh maka nilai variable index akan di tambah satu index++). Inilah mengapa di awal variable index di inisialisai dengan -1 dimana array masih kosong ketika menambah data variable index akan ditambah satu (index++) berarti variable index akan bernilai 0 dimana index array dimulai dari 0. Array stack index ke 0 akan diisi dengan data dari parameter fungsi ini (stack[index]=data).

```
void pop(){
    if(isEmpty() == 1){
        printf("Stack kosong \n");
    }else{
        printf("Data terambil: %d\n", stack[index]);
        index--;
    }
}
```

Fungsi ini untuk menghapus data dalam stack, dimana fungsi ini akan memeriksa menggunakan fungsi isEmpty apakah array kosong atau tidak. Jika array dalam keadaan kosong maka tidak ada data yang dapat dihapus. Sebaliknya jika array tak kosong maka fungsi ini akan menghapus data dengan mengurangi nilai variable index yang artinya index akan dikurangi dalam array.

```
void tampil(){
    for(int a=index; a>=0; a--){
        printf("%d \n", stack[a]);
    }
}
```

Fungsi tampil untuk mencetak data stack, dengan menggunakan perulangan yang dimulai dari index paling akhir ke 0 berulang sebanyak nilai dari variable index maka perulangan ini akan mencetak setiap data yang ada dalam array atau stack.

Implementasi stack dengan Linked List:

The screenshot shows a code editor window with a purple header bar containing the menu: File, Edit, Selection, View, Go, Run, Terminal, Help. On the left side, there is a vertical toolbar with icons for file operations like Open, Save, Find, Copy, Paste, and a gear icon.

```
C stack.c 1 ×
C stack.c > ...
1 #include<stdio.h>
2 #include<conio.h>
3 #include<malloc.h>
4
5 struct stack{
6     int angka;
7     struct stack *next;
8 }*head;
9
10 void push(int bilangan){
11     struct stack *baru, bantu;
12     baru = (struct stack*)malloc(sizeof(struct stack));
13     baru -> angka = bilangan;
14     if(head == NULL){
15         head = baru;
16         baru -> next = NULL;
17     }else{
18         baru -> next = head;
19         head = baru;
20     }printf("Data berhasil disimpan \n");
21 }
22
23 void pop(){
24     if(head == NULL){
25         printf("Stack kosong \n");
26     }else{
27         head = head -> next;
28         printf("Data berhasil dihapus \n");
29     }
30 }
31
32 void tampil(){
33     struct stack *bantu;
34     bantu = head;
35     printf("--- \n");
36     while(bantu!=NULL){
37         printf("|%d|\n", bantu -> angka);
38         printf("--- \n");
39         bantu = bantu -> next;
40     }
}
```

The screenshot shows a continuation of the code editor window. The code has been scrolled down to show the main function.

```
C stack.c 1 ×
C stack.c > ...
41 }
42
43 void main(){
44     push(0);
45     push(1);
46     push(2);
47     push(3);
48     push(4);
49     push(5);
50     push(6);
51     push(7);
52     push(8);
53     pop();
54     push(9);
55     push(10);
56     tampil();
57     getch();
58 }
```

Penjelasan:

```
struct stack{
    int angka;
    struct stack *next;
}*head;
```

Untuk membuat sebuah program stack dengan linked list maka kita harus membuat struct terlebih dahulu yang isinya variable angka dengan tipe data integer dan juga variable pointer next untuk menghubungkan antar data stack. Dan juga seperti biasa kita harus membuat variable pointer head sebagai kepala dari stack ini.

```
void push(int bilangan){
    struct stack *baru, bantu;
    baru = (struct stack*)malloc(sizeof(struct stack));
    baru -> angka = bilangan;
    if(head == NULL){
        head = baru;
        baru -> next = NULL;
    }else{
        baru -> next = head;
        head = baru;
    }printf("Data berhasil disimpan \n");
}
```

Fungsi push seperti bahasan di awal berfungsi untuk menambah data. Pertama kita harus membuat variabel dengan tipe struct stack untuk menampung data yang baru dengan nama variable baru. Kondisinya jika head masih kosong ini berarti kita tinggal menginisialisasi head dengan baru. Namun jika kondisinya head sudah terisi maka kita tinggal menempatkan baru didepan head dan mengganti baru menjadi head yang baru, coba ingat lagi prinsip tambah depan dalam bahasan linked list sebelumnya.

```
void pop(){
    if(head == NULL){
        printf("Stack kosong \n");
    }else{
        head = head -> next;
        printf("Data berhasil dihapus \n");
    }
}
```

Sedangkan fungsi pop yaitu untuk mengambil data. Kondisinya jika headnya bernilai NULL maka tidak akan terjadi penghapusan data karena datanya kosong. Jika sebaliknya maka head akan dilepas dari ikatan linked list, dengan cara memindahkan head ke node yang ada didepannya (head=head->next). Tapi pada hakikatnya data itu bukan dihapus tetapi hanya terlepas dari ikatan adi datanya masih ada dalam memori komputer maka dari itu ayo coba cari cara agar data benar-benar terhapus.

```
void tampil(){
    struct stack *bantu;
    bantu = head;
    printf("--- \n");
    while(bantu!=NULL){
        printf("|%d|\n", bantu -> angka);
        printf("--- \n");
        bantu = bantu -> next;
    }
}
```

Fungsi tampil digunakan untuk menampilkan data. Bagaimana fungsi ini dapat menampilkan data ? Sama seperti fungsi tampil sebelum-sebelumnya cara kerjanya sama. Yaitu dengan bantuan variable struct stack yang bernama bantu si bantu ini akan berjalan-jalan sampai kondisi bantunya bernilai NULL dia akan mengunjungi tiap node dan mencetak data dari tiap node tersebut.

QUEUE

Queue atau antrian adalah sekumpulan data dimana penambahan elemennya hanya bisa dilakukan pada satu ujung yang disebut sisi belakang (rear) dan penghapusannya dilakukan lewat ujung lain yaitu sisi depan (front). Queue atau antrian banyak ditemui dalam kehidupan sehari-hari. Misalnya, antrian mobil di gerbang tol; orang-orang yang mengantri untuk berobat dan masih banyak lagi.

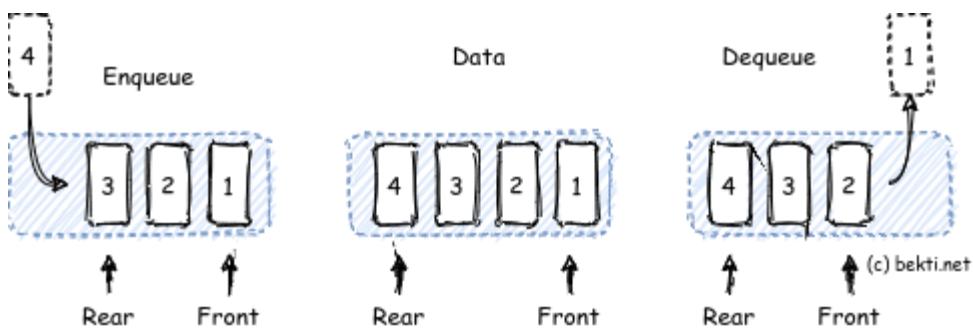
Jika pada stack menggunakan prinsip LIFO (Last In First Out). Maka pada queue menggunakan prinsip FIFO (First In First Out) yang artinya data yang pertama masuk ialah data yang akan di akses. Seperti konsep antrian maka yang pertama mengantri ialah yang akan pertama diproses.

Pada queue tidak ada batasan untuk banyaknya antrian tapi jika dalam pembuatannya kita menggunakan array maka kita harus membatasi banyaknya antrian. Ini dikarenakan array memiliki batasan (upperbound). Oleh karena itu jika queue tidak ingin dibatasi jumlahnya maka kita harus menggunakan linked list.

Operasi dasar yang terdapat pada Queue yaitu:

- a. Enqueue : berfungsi untuk menambahkan satu elemen ke dalam queue. Akibat dari fungsi ini jumlah elemen dalam queue akan bertambah.
- b. Dequeue : berfungsi untuk mengeluarkan atau mengambil satu elemen dari dalam queue. Elemen yang keluar dari dalam queue adalah yang berada pada posisi depan. Akibat dari operasi ini jumlah elemen dalam queue akan berkurang

Berikut ilustrasi Queue secara sederhana:



Queue dengan Array:

```

File Edit Selection View Go Run Terminal Help
C stack.c C queue.c 1 ×
C queue.c > dequeuel()
1 #include<stdio.h>
2 #include<conio.h>
3
4 int queue[10];
5 int index = -1;
6
7 int isEmpty(){
8     if(index == -1){
9         return 1;
10    }else{
11        return 0;
12    }
13 }
14
15 int isFull(){
16     if(index == 9){
17         return 1;
18     }else{
19         return 0;
20     }
21 }
22
23 void enqueue(int data){
24     if(isFull() == 1){
25         printf("Queue sudah penuh \n");
26     }else{
27         index++;
28         queue[index] = data;
29         printf("Data berhasil disimpan \n");
30     }
31 }
32
33 void dequeuel()
34 {
35     if(isEmpty() == 1){
36         printf("Queue kosong \n");
37     }else{
38         printf("Data terambil: %d\n", queue[0]);
39         for (int a=0; a<index; a++){
40             queue[a] = queue[a+1];
41         }
42         index--;
43     }
44 }
45 void tampil(){
46     for(int a=0; a<index; a++){
47         printf("%d ", queue[a]);
48     }
49 }

```

```
50 void main(){
51     enqueue(1);
52     enqueue(2);
53     enqueue(3);
54     dequeue();
55     tampil();
56     getch();
57 }
```

Penjelasan:

```
int queue[10];
int index = -1;
```

Pendeklarasian array yang queue dengan panjang 10. Ini digunakan untuk menampung data queue yang berarti data maksimum queue yang masuk adalah 10 data.

Pendeklarasian dan inisialisasi variable index yang bertipe integer dengan nilai -1. Ini digunakan untuk mengosongkan array atau sebagai tanda bahwa array masih belum diisi.

```
int isEmpty(){
    if(index == -1){
        return 1;
    }else{
        return 0;
}
```

Fungsi isEmpty untuk memeriksa apakah array queue kosong atau tidak. Jika kosong, fungsi mengembalikan nilai 1 dan jika tidak, fungsi mengembalikan nilai 0.

```
int isFull(){
    if(index == 9){
        return 1;
    }else{
        return 0;
}
```

Fungsi isFull untuk memeriksa apakah array queue penuh atau tidak. Jika penuh, fungsi mengembalikan nilai 1 dan jika tidak, fungsi mengembalikan nilai 0.

```
void enqueue(int data){
    if(isFull() == 1){
        printf("Queue sudah penuh \n");
    }else{
        index++;
        queue[index] = data;
        printf("Data berhasil disimpan \n");
    }
}
```

Fungsi enqueue berfungsi untuk menambah data. Kondisinya jika fungsi isFull mengembalikan nilai 1 berarti queue sudah penuh. Jika tidak maka masih dapat menambahkan data dengan cara Variabel index di increment, setelah itu array queue pada index ke index dinisialisasi oleh data.

```
void dequeue(){
    if(isEmpty() == 1){
        printf("Queue kosong \n");
    }else{
        printf("Data terambil: %d\n", queue[0]);
        for (int a=0; a<index; a++){
            queue[a] = queue[a+1];
        }
    }index--;
}
```

Fungsi dequeue berfungsi untuk mengeluarkan atau mengambil satu elemen dari dalam queue. Kondisinya jika fungsi isEmpty mengembalikan nilai 1 berarti queue kosong. Jika tidak, maka masih dapat mengambil data. Looping for digunakan untuk memindahkan data dari index n+1 ke n. Karena queue merupakan konsep antrian maka ketika data diambil maka yang terambil adalah data index ke nol. Agar data setelahnya pindah ke index data sebelumnya maka digunakan looping.

```
void tampil(){
    for(int a=0; a<=index; a++){
        printf("%d ", queue[a]);
    }
}
```

Fungsi tampil digunakan untuk menampilkan data queue. Looping untuk mencetak isi array queue. Looping dimulai dari nol sampai index. Looping akan berulang sebanyak index. Jika nilai a masih <=index maka program akan mencetak data array queue index ke a.

Queue dengan Linked list:

The image shows a screenshot of a code editor with a purple header bar containing the following menu items: File, Edit, Selection, View, Go, Run, Terminal, and Help. Below the header, there are two tabs: 'stack.c' and 'queue.c'. The 'queue.c' tab is currently active, indicated by a green background. On the left side of the editor, there is a vertical toolbar with several icons: a file icon, a search icon, a refresh icon, a file navigation icon, and a zoom icon. The main workspace contains the following C code:

```
C queue.c > ...
1 #include<stdio.h>
2 #include<conio.h>
3 #include<malloc.h>
4
5 struct queue{
6     int angka;
7     struct queue *next;
8 }*head=NULL;
9
10 void enqueue(int data){
11     struct queue *baru, *bantu;
12     baru = (struct queue*)malloc(sizeof(struct queue));
13     baru->angka = data;
14     baru->next = NULL;
15     bantu = head;
16     if(head == NULL){
17         head = baru;
18     }else{
19         while(bantu->next!=NULL){
20             bantu = bantu->next;
21         }
22         bantu->next = baru;
23     }
24 }
25 void dequeue(){
26     if(head == NULL){
27         printf("Queue kosong");
28     }else if(head ->next == NULL){
29         head = NULL;
30     }else{
31         head = head -> next;
32     }
33 }
```

```

34
35 void tampil(){
36     struct queue *bantu;
37     bantu = head;
38     if(bantu != NULL){
39         printf("Elemen queue: \n");
40         while(bantu != NULL){
41             printf("|%d| ", bantu->angka);
42             bantu = bantu->next;
43         }
44     }else{
45         printf("Queue kosong");
46     }
47 }
48
49 void main(){
50     enqueue(1);
51     enqueue(2);
52     enqueue(3);
53     dequeue();
54     tampil();
55     getch();
56 }
```

Penjelasan:

```

struct queue{
    int angka;
    struct queue *next;
}*head=NULL;
```

Sama seperti membuat stack dengan linked list, dalam membuat queue dengan linked list juga kita harus membuat struct terlebih dahulu yang berisi variable angka dengan tipe data integer dan juga variable pointer next untuk menghubungkan antar data queue. Dan juga seperti biasa kita harus membuat variable pointer head sebagai kepala dari queue ini.

```

void enqueue(int data){
    struct queue *baru, *bantu;
    baru = (struct queue*)malloc(sizeof(struct queue));
    baru->angka = data;
    baru->next = NULL;
    bantu = head;
    if(head == NULL){
        head = baru;
    }else{
        while(bantu->next!=NULL){
            bantu = bantu->next;
        }bantu->next = baru;
    }
}
```

Didalam fungsi enqueue yang digunakan untuk menambah data berisi kondisi jika head masih kosong maka kita tinggal menginisialisasi head dengan baru. Sedangkan jika head sudah terisi maka data akan

ditambahkan dibelakang dengan menggunakan perulangan. Looping while akan terus berulang selama data setelah bantu tidak sama dengan NULL. Kemudian bantu diisi oleh data dari bantu ke selanjutnya.

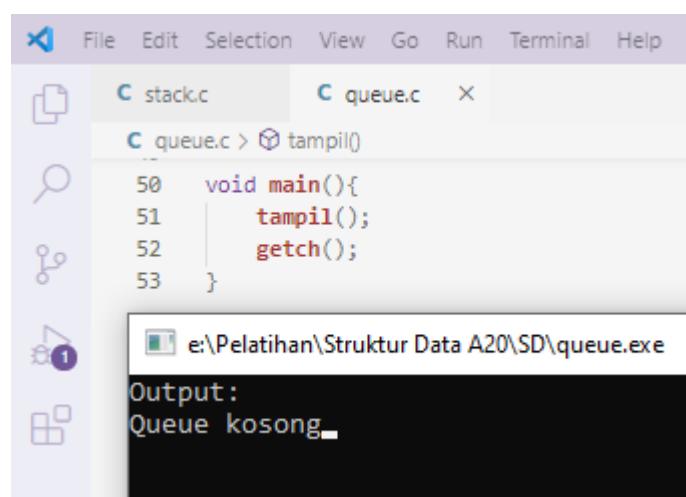
```
void dequeue(){
    if(head == NULL){
        printf("Queue kosong");
    }else if(head ->next == NULL){
        head = NULL;
    }else{
        head = head -> next;
    }
}
```

Dalam fungsi dequeue yang digunakan untuk mengambil data, berisi kondisi jika head sama dengan NULL maka queue kosong, jika head ke data selanjutnya sama dengan NULL maka inisialisasi head dengan NULL, sedangkan selain dua kondisi itu maka head dipindah ke data selanjutnya setelah head.

```
void tampil(){
    struct queue *bantu;
    bantu = head;
    if(bantu != NULL){
        printf("Elemen queue: \n");
        while(bantu != NULL){
            printf("%d ", bantu->angka);
            bantu = bantu->next;
        }
    }else{
        printf("Queue kosong");
    }
}
```

Fungsi tampil digunakan untuk menampilkan data. Sama seperti fungsi tampil sebelum-sebelumnya cara kerjanya sama. Yaitu dengan bantuan variable struct queue yang bernama bantu si bantu ini akan berjalan-jalan sampai kondisi bantunya bernilai NULL dia akan mengunjungi tiap node dan mencetak data dari tiap node tersebut.

Ketika program dijalankan:



```
File Edit Selection View Go Run Terminal Help
stack.c queue.c ×
queue.c > ⌂ tampil()
50 void main(){
51     tampil();
52     getch();
53 }

e:\Pelatihan\Struktur Data A20\SD\queue.exe
Output:
Queue kosong
```

```
File Edit Selection View Go Run Terminal Help
C stack.c C queue.c X
C queue.c > tampil()
50 void main(){
51     enqueue(1);
52     enqueue(2);
53     enqueue(3);
54     dequeue();
55     tampil();
56     getch();
57 }

e:\Pelatihan\Struktur Data A20\SD\queue.exe
Output:
Elemen queue:
|2| |3| -
```

Ketika user menambah data (enqueue) 1, 2, dan 3 maka data queue berisi:

| 1 | - | 2 | - | 3 |

Ketika user mengambil data (dequeue) maka data yang paling awal masuk akan terambil sehingga menjadi:

| 2 | - | 3 |

SORTING

1) Pengertian Sorting

Sorting (pengurutan) adalah proses pengurutan data yang sebelumnya disusun secara acak atau tidak teratur menjadi tersusun secara teratur menurut aturan tertentu. Sorting disebut juga sebagai salah satu algoritma untuk meletakkan kumpulan elemen data ke dalam urutan tertentu berdasarkan satu atau beberapa kunci dalam tiap-tiap elemen. Pengurutan data dalam struktur data sangat penting terutama untuk data yang bertipe data numerik ataupun karakter.

Pada dasarnya ada dua macam urutan yang biasa digunakan dalam suatu proses sorting:

1. Urut naik (*ascending*) Mengurutkan dari data yang mempunyai nilai paling kecil sampai paling besar.
2. Urut turun (*descending*) Mengurutkan dari data yang mempunyai nilai paling besar sampai paling kecil.

2) Kegunaan Sorting

Tujuan utama dari proses sorting adalah untuk mengurutkan data, baik itu dari terendah ataupun tertinggi. Yang secara tidak langsung menjadikan data lebih terstruktur, rapi dan teratur sehingga akses terhadap data tersebut akan menjadi lebih mudah dan efisien.

Ada banyak alasan dan keuntungan dengan mengurutkan data. Data yang terurut mudah untuk dicari, mudah untuk diperiksa, dan mudah untuk dibetulkan jika terdapat kesalahan. Data yang terurut dengan baik juga mudah untuk dihapus jika sewaktu-waktu data tersebut tidak diperlukan lagi. Selain itu, dengan mengurutkan data maka kita semakin mudah untuk menyisipkan data ataupun melakukan penggabungan data.

3) Metode-Metode Sorting

Proses pengurutan data (sorting) sering kali menjadi bagian yang krusial dalam setiap pengolahan data, contohnya dalam berbagai aplikasi dengan data ribuan bahkan jutaan, pengurutan data (sorting) dapat menghabiskan CPU-time paling banyak dari keseluruhan pengolahan data.

Masalah pengurutan itu sendiri selama ini memunculkan berbagai macam variasi algoritma. Algoritma-algoritma tersebut di antaranya, adalah algoritma berdasarkan priority queue (Selection Sort dan Heap Sort), algoritma penyisipan dalam keterurutan (Insertion Sort dan Tree Sort), algoritma transposisi (Bubble Sort), algoritma dengan increment yang mengecil (Shell Sort), algoritma divide and conquer (Merge Sort dan Quick Sort) dan algoritma penghitungan alamat (Radix Sort dan proximity map sort). Di luar algoritma-algoritma tersebut masih ada puluhan algoritma lain yang pada dasarnya hanya merupakan varian-varian dari algoritma-algoritma itu sendiri.

Tetapi dari sekian banyaknya Algoritma Sorting, yang akan kita pelajari adalah:

1. Metode Bubble Sort
2. Metode Selection Sort
3. Metode Insertion Sort
4. Metode Counting Sort
5. Metode Quick Sort

6. Metode Merge Sort

1) Metode Bubble Sort

Metode pengurutan gelembung (Bubble Sort) diinspirasikan oleh gelembung sabun yang berada di permukaan air. Karena berat jenis gelembung sabun lebih ringan daripada berat jenis air, maka gelembung sabun selalu terapung ke atas permukaan. Prinsip di atas dipakai pada pengurutan gelembung.

Bubble sort (pengurutan gelembung) merupakan metode sorting dengan cara menukar atau membandingkan dua buah data yang bersebelahan, yaitu antara data pertama dan data kedua, data kedua dan data ketiga, begitu seterusnya berulang sampai data terakhir. Pengulangan tersebut akan terus terjadi sebanyak n buah data. Disebut bubble sort karena proses pengurutan secara berangsur-angsur bergerak/berpindah ke posisinya yang tepat, seperti gelembung yang mengapung ke atas permukaan.

Algoritma Bubble Sort

- a. Algoritma dimulai dari elemen paling awal.
- b. Dua buah elemen pertama dari list dibandingkan.
- c. Jika elemen pertama lebih besar dari elemen kedua, dilakukan pertukaran. Ini jika ascending, jika descending maka sebaliknya.
- d. Langkah 2 dan 3 dilakukan lagi terhadap elemen kedua dan ketiga, seterusnya sampai ke ujung elemen.
- e. Bila sudah sampai ke ujung dilakukan lagi ke awal sampai tidak ada terjadi lagi pertukaran elemen.
- f. Bila tidak ada pertukaran elemen lagi, maka list elemen sudah terurut.

Agar lebih jelas, mari kita lihat ilustrasi Bubble Sort berikut ini:



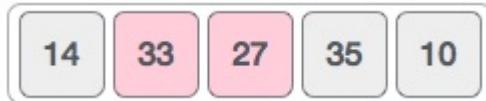
Anggap kita mempunyai 5 elemen array seperti di atas



Proses pertama, kita akan membandingkan elemen pertama (14) dengan elemen kedua (33)



Elemen pertama dan kedua ternyata sudah terurut, maka kita akan melewatkannya dan melanjutkan melakukan perbandingan antara elemen kedua (33) dengan elemen ketiga (27)



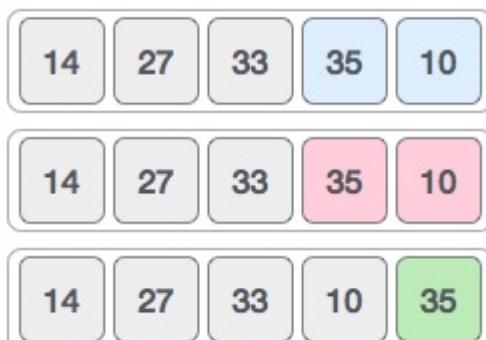
Ternyata elemen kedua lebih besar dari elemen ketiga, maka kita akan menukar (swap) kedua elemen tersebut



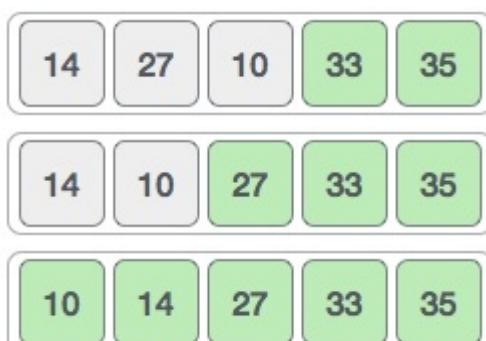
Elemen kedua dan ketiga sudah berhasil kita urut, maka kita akan melanjutkan ke proses selanjutnya



Proses perbandingan 2 elemen akan terus berlanjut



Proses perbandingan telah sampai di elemen paling akhir yang ada di dalam array, tetapi data kita belum semuanya terurut, maka proses perbandingan akan terjadi lagi dimulai dari awal (elemen pertama)



Apabila semua elemen sudah terurut sebagaimana mestinya dan tidak ada proses pertukaran (swap) yang terjadi lagi, maka hal ini akan memberitahu komputer bahwa semua elemen kita sudah berhasil diurutkan.

Kelebihan Bubble Sort

- Proses penghitungan Bubble Sort merupakan metode yang paling sederhana.
- Algoritma Bubble Sort mudah dipahami

- c. Langkah atau tahapan dalam pengurutan data sangat sederhana

Kelemahan Bubble Sort

- a. Proses penghitungan Bubble Sort menggunakan metode pengurutan termasuk paling tidak efisien walaupun dianggap sederhana. Karena proses pengurutan data dilakukan dengan tahapan satu-satu, mulai dari data paling awal sebelah kiri sampai data terakhir.
- b. Pada saat mengurutkan data dalam jumlah yang besar, maka proses penghitungan akan semakin lama dan melambat. Hal ini dikarenakan Bubble sort menggunakan proses pengurutan data secara tunggal (satu-satu).
- c. Jumlah pengulangan akan tetap sama sampai data yang terakhir, walaupun sebagian data yang ada telah terurut.

Implementasi dalam Program

```
#include<stdio.h>

void bubbleSortAsc(int data[], int n){
    // Melakukan perulangan sebanyak n-1, karena index dimulai dr 0
    for(int i =0; i<n-1; i++){

        //Melakukan looping untuk mengakses array pada setiap data.
        for(int j=0; j<n-1-i; j++){

            //jika data index j lebih besar dari data index j+1
            if(data[j] > data[j+1]){

                // Maka, tukar (swap) nilai arr[j] dengan nilai arr[j+1]
                int temp = data[j];
                data[j]=data[j+1];
                data[j+1]=temp;
            }
        }
    }
}
```

```

void cetak(int arr[], int n){
    for(int i=0; i<n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main(){
    int data[] = {14, 33, 27, 35, 10};

    //Mencari panjang array dan menyesuaikan ukuran array dari data tersebut.
    int n = sizeof(data)/sizeof(data[0]);

    printf("Sebelum Sort: ");
    cetak(data,n);

    printf("Sesudah Sort: ");
    bubbleSortAsc(data, n);
    cetak(data, n);

    return 0;
}

```

LATIHAN!

1. Buatlah proses sorting menggunakan Bubble Sort dari array di bawah ini!

42	13	8	-12	3	59	17	3
----	----	---	-----	---	----	----	---

2. Buatlah program bubble sort untuk pengurutan data secara descending dengan data yang diinputkan!

2) Metode Selection Sort

Selection Sort merupakan salah satu *algoritma sorting* yang **populer** dan cukup sering digunakan. Sesuai namanya, **algoritma sorting** yang satu ini akan melakukan penyeleksian data dari kumpulan data yang belum terurut lalu setiap data ditempatkan pada posisinya masing-masing. Sudah jelas, dalam tahap penyeleksianya juga terjadi **searching data** selanjutnya yang akan di tempatkan pada posisinya yang tepat.

Salah satu bentuk implementasinya adalah saat melakukan baris berbaris dari siswa yang **terpendek** hingga **tertinggi**. Pasti kita akan mencari yang terpendek lalu siswa tersebut masuk ke barisan. Dilanjutkan lagi dengan siswa yang **kedua terpendek**, memasuki barisan. Begitu selanjutnya.

Untuk mencari siswa terpendek, dilakukan **searching**. Oleh karena itu, algoritma selection sort juga disebut sebagai algoritma hasil penggabungan **sorting dan searching**.

Algoritma Selection Sort

Algoritma Selection Sort dapat dirangkum sebagai berikut:

1. Temukan nilai yang paling terkecil dari data pertama sampai data terakhir, kemudian ditukar posisinya dengan data pertama. Ini jika ascending, jika descending temukan nilai data yang terbesar.
2. Temukan nilai data terkecil dari data kedua sampai data terakhir, kemudian ditukar posisinya dengan data kedua.
3. Temukan nilai data terkecil dari data ketiga sampai data terakhir, kemudian ditukar posisinya dengan data ketiga.
4. Lakukan seterusnya sampai semua data urut naik. Apabila terdapat n buah data yang akan diurutkan, maka membutuhkan $n-1$ langkah pengurutan, dimana data terakhir yaitu data ke- n tidak perlu diurutkan karena hanya tinggal satu-satunya.

Agar semakin paham ayo kita lihat ilustrasi Selection Sort berikut ini:



Anggaplah kita mempunyai 8 elemen array yang masih tidak berurutan seperti ini



Selection sort akan mencari elemen terkecil dari array yang kita punya, dimana disini elemen terkecil adalah 10



Setelah berhasil mendapatkan elemen terkecil, maka selection sort ini akan menukar elemen tersebut ke posisi pertama (10 dan 14 ditukar)



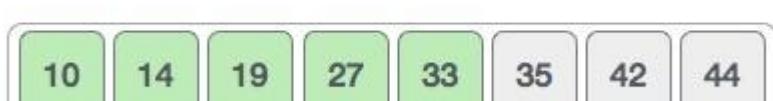
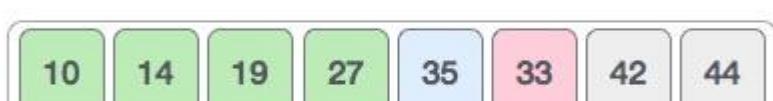
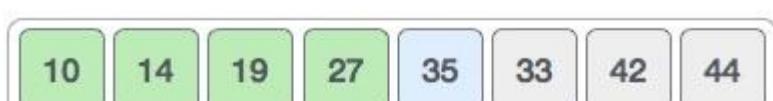
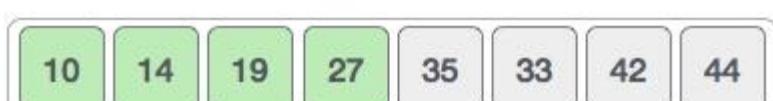
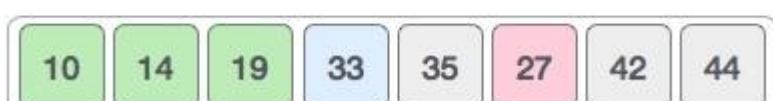
Proses pengurutan pertama berhasil dilakukan, maka selection sort ini akan mengulangi lagi hal yang sama, yaitu mencari elemen terkecil yang ada di dalam array kita



Ternyata 14 merupakan elemen terkecil yang ada di dalam array, maka selection sort akan menukar posisi elemen 14 ke posisi kedua (mengingat kita sudah berhasil meletakan elemen terkecil di posisi pertama)



Elemen 14 sudah berhasil kita tukar ke posisi kedua, maka dengan itu proses pengurutan kedua berhasil kita lakukan dan selection sort akan melanjutkannya ke proses pengurutan selanjutnya



Proses tersebut akan dilakukan secara lagi dan lagi sampai semua elemen array berhasil kita sorting.

Kelebihan Selection Sort:

- Algoritma ini sangat rapat dan mudah untuk diimplementasikan
- Mempercepat pencarian

- Mudah menentukan data maksimum /minimum.
- Mudah menggabungkannya kembali.
- Kompleksitas selection sort relatif lebih kecil.

Kekurangan Selection Sort:

- Membutuhkan method tambahan
- Sulit untuk mengatasi masalah
- Perlu dihindari untuk penggunaan data lebih dari 1000 tabel, karena akan menyebabkan kompleksitas yang lebih tinggi dan kurang praktis

```
#include <stdio.h>

void selectionSortAsc (int data[], int n) {
    // variabel sementara untuk menyimpan index data terkecil
    int min;

    // Melakukan perulangan sebanyak n-1, karena index dimulai dr 0
    for(int i = 0; i < n-1 ; i++)  {

        // anggap saja bahwa index ke i adalah data terkecil
        min = i ;
        for(int j = i+1; j < n ; j++ ) {

            // jika data index j lebih kecil dari data index min
            if(data[ j ] < data[ min ])  {

                // maka, ganti nilai min menjadi nilai j
                min = j ;
            }
        }

        //menukar nilai (swap)
        int temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
}
```

```

void cetak(int data[], int n){
    for (int i = 0; i < n; i++){
        printf("%d ", data[i]);
    }
    printf("\n");
}

int main(){
    int data[] = {14, 33, 27, 10, 35, 19, 42, 44};

    //Mencari panjang array dan menyesuaikan ukuran array dari data tersebut.
    int n = sizeof(data)/sizeof(data[0]);

    printf("Sebelum Sort: ");
    cetak(data, n);

    printf("Sesudah Sort: ");
    selectionSortAsc(data, n);
    cetak(data, n);

    return 0;
}

```

LATIHAN!

1. Buatlah proses sorting menggunakan metode selection sort dari array di bawah ini!

20	0	9	31	12	-7	10	2
----	---	---	----	----	----	----	---

2. Buatlah program selection sort untuk pengurutan data secara descending dengan data yang diinputkan!

3) Metode Insertion Sort

Algoritma insertion sort adalah sebuah algoritma sederhana yang cukup efisien untuk mengurutkan sebuah list yang hampir terurut. Algoritma ini juga bisa digunakan sebagai bagian dari algoritma yang lebih canggih. Cara kerja algoritma ini adalah dengan mengambil elemen list satu-per-satu dan memasukkannya di posisi yang benar seperti namanya.

Pada array, list yang baru dan elemen sisanya dapat berbagi tempat di array, meskipun cukup rumit. Untuk menghemat memori, implementasinya menggunakan pengurutan di tempat yang membandingkan elemen saat itu dengan elemen sebelumnya yang sudah diurut, lalu menukar posisinya terus sampai posisinya tepat. Hal ini terus dilakukan sampai tidak ada elemen tersisa yang di input.

Salah satu implementasinya pada kehidupan sehari-hari adalah saat kita mengurutkan kartu remi. Pertama-tama Anda meletakkan kartu-kartu tersebut di atas meja, kemudian melihatnya dari kiri ke kanan. Apabila kartu di sebelah kanan lebih kecil daripada kartu di sebelah kiri, maka ambil kartu tersebut dan sisipkan di tempat yang sesuai.

Algoritma Insertion Sort

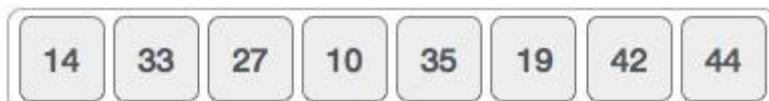
Penganalogian Insertion Sort dengan pengurutan kartu. Berikut menjelaskan bagaimana algoritma Insertion Sort bekerja dalam pengurutan kartu, anggaplah kita ingin mengurutkan satu set kartu dari kartu yang bernilai paling kecil hingga yang paling besar.

1. Dimulai dengan posisi tangan kosong, dan semua kartu berada di atas meja. Dan anggaplah kita akan menyusun kartu ke tangan kiri kita.
2. Mengambil kartu pertama dari meja dan meletakkannya ke tangan kiri.
3. Mengambil kartu kedua dan membandingkannya dengan kartu yang sudah ada di tangan kiri.
4. Jika kartu yang diambil dari meja memenuhi syarat perbandingan, maka kartu tersebut akan diletakan di depan kartu yang dibandingkan, serta kartu yang lain yang telah dibandingkan akan bergeser mundur (ke belakang).

Proses ini akan berlangsung sampai semua kartu akan terurutkan dengan benar sesuai kriteria pengurutannya. Demikian juga halnya dalam pengurutan data.

Jika data sudah ada, maka pengurutan dimulai dengan mengambil satu data dan membandingkannya dengan data-data yang ada di depannya. Jika data yang diambil memenuhi syarat perbandingan, maka data yang diambil tersebut akan diletakan di depan data yang dibandingkan, kemudian data-data yang dibandingkan akan bergeser mundur.

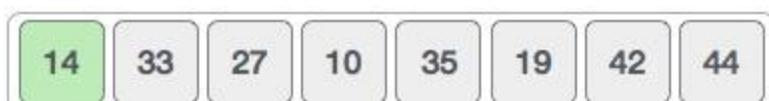
Catatan: Dalam hal pengurutan data dengan metode insertion sort ini, data yang diambil pertama adalah data kedua, kemudian data yang diambil akan dibandingkan dengan data – data yang ada di sebelah kiri / data sebelumnya (data-data sebelum data yang diambil). Jika proses tersebut selesai, maka akan dilanjutkan dengan data-data selanjutnya (data ke-3, data ke-4... dan seterusnya). Proses akan berlangsung sampai data – data terurutkan dengan benar.



Anggap kita mempunya list elemen array seperti ini



Proses pertama, kita membandingkan elemen pertama (14) dengan elemen kedua (33)



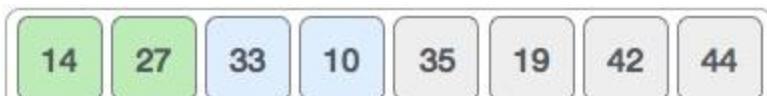
Hasilnya ternyata elemen pertama kita (14) dan elemen kedua (33) sudah terurut dengan benar



Selanjutnya kita lanjutkan ke proses kedua, di mana kita membandingkan elemen kedua (33) dengan elemen ketiga (27)



Setelah kita bandingkan ternyata elemen ketiga kita (27) lebih kecil dari elemen kedua (33) kita, lalu kita harus menukarnya (swap). Di proses ini, kita tidak akan lanjut membandingkan elemen ketiga (33) dengan keempat (10), sebagai gantinya kita akan membandingkan elemen pertama (14) dan elemen yang baru kita urut yaitu kedua (27). Ternyata setelah dibandingkan kedua elemen kita tersebut sudah terurut dengan benar, maka barulah algoritma ini akan melanjutnya proses membandingkan elemen ketiga dan keempat.



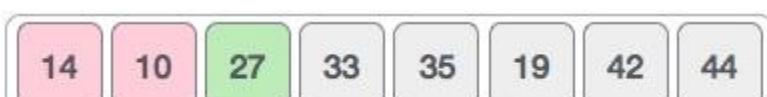
Elemen ketiga (33) ternyata lebih besar dibandingkan elemen keempat (10) maka kita harus menukarnya (swap)



Selanjutnya kita akan membandingkan elemen yang baru kita tukar/urut (10) dengan elemen terakhir yang sudah kita urutkan (27)



Ternyata elemen yang baru kita tukar/urut (10) lebih kecil dibandingkan elemen terakhir yang sudah kita urutkan (27), maka kita harus menukarnya



Selanjutnya kita akan membandingkan lagi elemen yang baru kita tukar/urut dengan elemen terakhir yang sudah kita urutkan (14)



Setelah dibandingkan ternyata kedua elemen tersebut harus ditukar. Proses seperti ini akan terus berjalan sampai benar-benar tidak ada lagi elemen yang harus ditukar posisinya

Kelebihan Insertion Sort

- Sederhana dalam penerapannya.
- Efisien dalam data yang kecil.
- Jika list sudah terurut atau sebagian terurut maka Insertion Sort akan lebih cepat dibandingkan dengan Quicksort.
- Efisien dalam data yang sebagian sudah terurut.
- Lebih efisien dibanding Bubble Sort dan Selection Sort.
- Loop dalam pada Inserion Sort sangat cepat, sehingga membuatnya salah satu algoritma pengurutan tercepat pada jumlah elemen yang sedikit.
- Stabil.

Kekurangan Insertion Sort

- Banyaknya operasi yang diperlukan dalam mencari posisi yang tepat untuk elemen larik.
- Untuk larik yang jumlahnya besar ini tidak praktis.
- Jika list terurut terbalik sehingga setiap eksekusi dari perintah harus memindai dan mengganti seluruh bagian sebelum menyisipkan elemen berikutnya.
- Membutuhkan waktu $O(n^2)$ pada data yang tidak terurut, sehingga tidak cocok dalam pengurutan elemen dalam jumlah besar.

```
#include<stdio.h>

void insertionSort(int data[], int n){
    int temp, j;

    // Melakukan perulangan sebanyak n-1, karena index dimulai dr 0
    for (int i = 1; i <= n-1; i++){
        temp = data[i];
        j = i-1;

        //data[j] > temp untuk mengurutkan data secara ascending.
        //data[j] < temp untuk mengurutkan data secara descending.
        while(j>=0 && data[j] > temp){
            data[j+1] = data[j];
            j--;
        }
        data[j+1] = temp;
    }
}
```

```

void cetak(int data[], int n){
    for (int i = 0; i < n; i++){
        printf("%d ", data[i]);
    }
    printf("\n");
}

int main(){
    int data[] = {14, 33, 27, 10, 35, 19, 42, 44};

    //Mencari panjang array dan menyesuaikan ukuran array dari data tersebut.
    int n = sizeof(data)/sizeof(data[0]);

    printf("Sebelum Sort: ");
    cetak(data, n);

    printf("Sesudah Sort: ");
    insertionSort(data, n);
    cetak(data, n);

    return 0;
}

```

LATIHAN!

- Buatlah proses sorting menggunakan Insertion Sort dari array di bawah ini!

42	13	8	-12	3	59	17	3
----	----	---	-----	---	----	----	---

- Buatlah program Insertion sort untuk pengurutan data secara descending dengan data yang diinputkan!

Quick Sort

Algoritma quick sort diperkenalkan pertama kali oleh **C.A.R. Hoare** pada tahun **1960**, dan dimuat sebagai artikel di **"Computer Journal 5"** pada April **1962**. Quick sort adalah algoritma sorting yang berdasarkan pembandingan dengan metode divide-and-conqueror. Disebut Quick Sort, karena Algoritma quick sort mengurutkan dengan sangat cepat.

Algoritma quick sort mengurutkan dengan sangat cepat, namun algoritma ini sangat kompleks dan diproses secara rekursif. Sangat memungkinkan untuk menulis algoritma yang lebih cepat untuk beberapa kasus khusus, namun untuk kasus umum, sampai saat ini tidak ada yang lebih cepat dibandingkan algoritma quick sort.

Quick sort merupakan suatu algoritma pengurutan data yang menggunakan teknik pemecahan data menjadi partisi-partisi, sehingga metode ini disebut juga dengan nama partition exchange sort. Untuk memulai iterasi pengurutan, pertama-tama sebuah elemen dipilih dari data, kemudian elemen-elemen data akan diurutkan diatur sedemikian rupa.

Algoritma Quicksort

1. Divide and Conquer

Divide and conquer adalah metode pemecahan masalah yang bekerja dengan membagi (*divide*) masalah menjadi beberapa sub-masalah yang sama atau berhubungan, hingga masalah tersebut menjadi sederhana untuk dipecahkan (*conquer*) secara langsung. Pemecahan dari tiap-tiap sub-masalah kemudian digabungkan untuk memberikan solusi terhadap masalah semula.

Metode *divide and conquer* menawarkan penyederhanaan masalah dengan pendekatan tiga langkah sederhana: pembagian masalah menjadi sekecil mungkin, penyelesaian masalah-masalah yang telah dikecilkan, kemudian digabungkan kembali untuk mendapat solusi optimal secara keseluruhan. Khusus untuk *quicksort*, proses penggabungan (*combine*) tidak perlu dilakukan, karena sudah terjadi secara alami.

Prinsip dalam algoritma *quicksort* sebagai berikut (diuraikan pula oleh Sedgewick):

- 1 Bila elemen dalam array kurang dari jumlah tertentu (biasanya 2), proses selesai.
- 2 Ambil sebuah elemen yang berfungsi sebagai poros.
- 3 Pisahkan array dalam 2 bagian, sebelah kiri lebih kecil dari poros, sebelah kanan lebih besar dari poros.
- 4 Ulangi proses secara rekursif pada tiap-tiap bagian.

Hal penting dari hal algoritma ini adalah: bagaimana memilih poros dengan tepat dan secara efisien mengatur tiap-tiap elemen sehingga didapat elemen kecil > poros > elemen besar dalam kondisi (mendekati) seimbang.



1 2 5 7 3 14 7 26 12 run quick sort recursively

...

1 2 3 5 7 7 12 14 26 sorted

Cara mengurutkan data dengan Quick Sort:

Pertama siapkan angka yang ingin diurutkan dengan metode Quick Sort

1 12 5 26 7 14 3 7 2 unsorted

Pilih elemen sementara atau pivot value, contohnya 7, elemen sementara ini berguna sebagai patokan selesainya partisi. Karena hanya sementara, jadi kalau partisinya sudah mencapai elemen sementara tersebut maka partisi selesai dan harus memilih elemen sementara yang lain sebagai patokan.



Setelah ditentukan elemen sementaranya, kita pilih angka pertama dan terakhir. Karena angka pertama sudah benar posisinya yaitu angka 1 lebih kecil dari angka 7 (elemen sementara), maka kita geser 1 angka dari angka pertama yaitu angka 12. Angka terakhir belum benar dan mesti dicocokkan jadi tidak harus menggeser ke angka sebelahnya terlebih dahulu.

Lalu setelah itu, angka 12 dan angka 2 kita tukar karena 2 lebih kecil dari angka elemen sementara tadi yaitu 7 dan angka 12 lebih besar dari angka 7, yang lebih kecil dari elemen dasar letaknya di kiri dan yang lebih besar di kanan.



Dilanjutkan dengan angka 5 dan 7,karena angka 5 sudah benar letaknya karena lebih kecil dari 7 (elemen dasar) maka geser 1 angka ke angka 26, karena angka 7 sama dengan 7 (elemen dasar) maka tetap.

Lalu tukar angka 26 dengan angka 7



Kemudian giliran angka 7 (Elemen sementaranya) dengan angka 3,caranya masih sama dengan yang tadi karena 3 lebih kecil dari 7 dan 7 lebih besar dari 3 jadi ditukar



Karena sudah mencapai elemen sementara tadi yaitu 7,maka partisi dengan elemen sementara angka 7 tadi selesai dan mendapatkan hasil seperti ini



Ulangi seperti langkah-langkah sebelumnya dengan elemen sementara yang baru sampai angka-angkanya terurut.



...

Jadilah urutan angka yang benar



Kelebihan Algoritma Quick Sort

Beberapa hal yang membuat quick sort unggul:

- Secara umum memiliki kompleksitas $O(n \log n)$.
- Algoritmanya sederhana dan mudah diterapkan pada berbagai bahasa pemrograman dan arsitektur mesin secara efisien.
- Dalam praktiknya adalah yang tercepat dari berbagai algoritma pengurutan dengan perbandingan, seperti merge sort dan heap sort.
- Melakukan proses langsung pada input (in-place) dengan sedikit tambahan memori.
- Bekerja dengan baik pada berbagai jenis input data (seperti angka dan karakter).

Kekurangan Algoritma Quick Sort

- Sedikit kesalahan dalam penulisan program membuatnya bekerja tidak beraturan (hasilnya tidak benar atau tidak pernah selesai).
- Memiliki ketergantungan terhadap data yang dimasukkan, yang dalam kasus terburuk memiliki kompleksitas $O(n^2)$.
- Secara umum bersifat tidak stable, yaitu mengubah urutan input dalam hasil akhirnya (dalam hal inputnya bernilai sama).
- Pada penerapan secara rekursif (memanggil dirinya sendiri) bila terjadi kasus terburuk dapat menghabiskan stack dan memacetkan program.

Implementasi Program

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

void quickSort (int a[],int lo,int hi[]){
    int i=lo, j=hi, h ;
    int pivot = a[lo];

    do{
        while (a[i]<pivot){
            i++;
        }
        while(a[j]>pivot){
            j--;
        }
        if(i<=j){
            h = a[i];
            a[i]=a[j];
            a[j]=h;
            i++;
            j--;
        }
    }while (i<=j);
    if(lo<j){
        quickSort(a,lo,j);
    }if(i<hi){
        quickSort(a,i,hi);
    }
}

void main(){
    int tabInt[10] = {1,12,5,26,7,14,3,7,2};
    int n=10;
    printf("Sebelum di sorting :\n");
    for(int i=0;i<n;i++){
        printf("%d ",tabInt[i]);
    }
    quickSort(tabInt,0,n-1);
    printf("\nSetelah di sorting :\n");
    for(int i=0;i<n;i++){
        printf("%d ",tabInt[i]);
    }
    getch();
}
```

1. Lakukan proses pengurutan data di atas secara ascending menggunakan metode quicksort
2. Lakukan proses pengurutan data di atas secara descending menggunakan metode quicksort

Metode Counting Sort

Algoritma Counting Sort

Counting Sort adalah algoritma pengurutan efektif dan efisien yang melakukan pengurutan dengan ide dasar meletakkan elemen pada posisi yang benar, dimana penghitungan posisi yang benar dilakukan dengan cara menghitung (counting) elemen-elemen dengan nilai lebih kecil atau sama dengan elemen tersebut. Contoh sederhana saja jika terdapat 12 elemen yang lebih kecil daripada x, maka x akan mendapatkan posisinya di posisi 13.



Tentu saja, sedikit modifikasi harus dilakukan agar metode ini dapat menangani kasus di mana terdapat elemen-elemen lain yang nilainya sama dengan x . Dimana tentu saja kita tidak dapat menempatkan semua elemen yang nilainya sama dengan x di posisi yang sama.

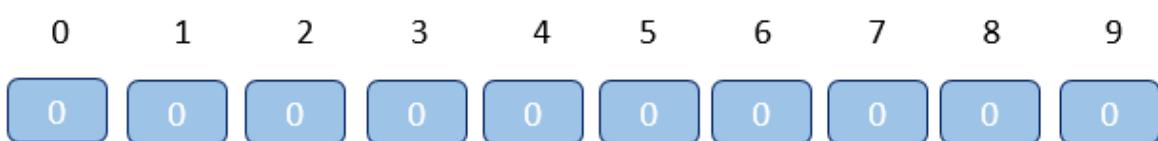
Misalkan ada array yang bernama `myArray` dengan data sebagai berikut:



Buat sebuah array baru untuk menyimpan hasil perhitungan setiap nilai yang unik yaitu `countArray`

Inisialisasi setiap index `countArray` tersebut dengan nilai 0

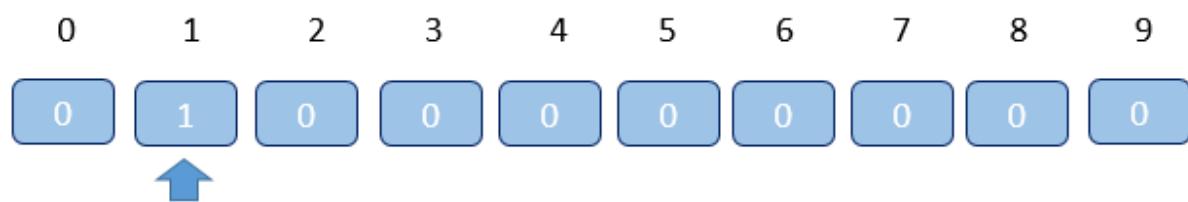
Index:



Lihat setiap element yang ada di `myArray`, elemen pertama adalah 1 itu berarti elemen 1 jumlahnya ada 1 lalu tempatkan jumlah tersebut di index 1 `countArray`



Index:

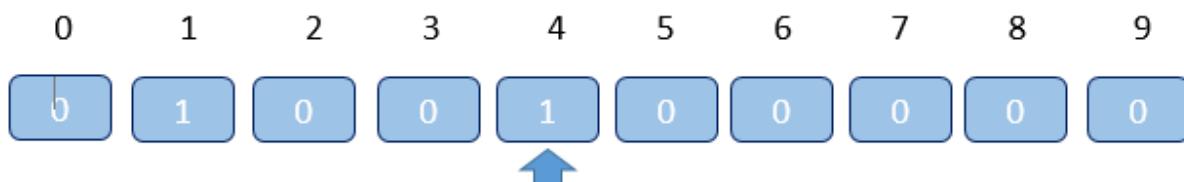


Sekarang elemen ke dua di `myArray` adalah empat

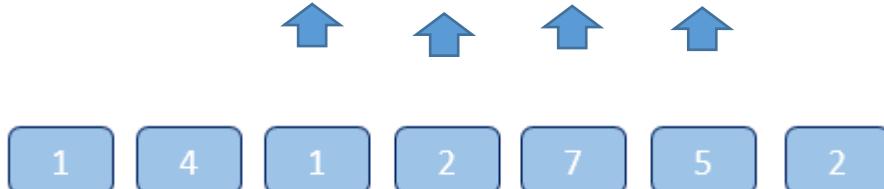


Maka index ke empat di `countArray` bertambah nilainya 1

Index:

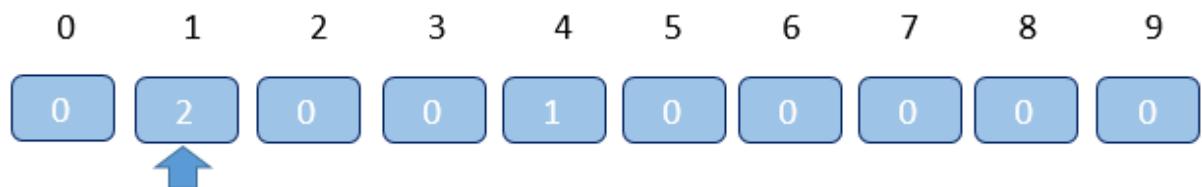


Sekarang elemen ke tiga di `myArray` yaitu 1



Maka index ke 1 bertambah lagi nilainya 1adi sekarang jumlahnya 2

Index:

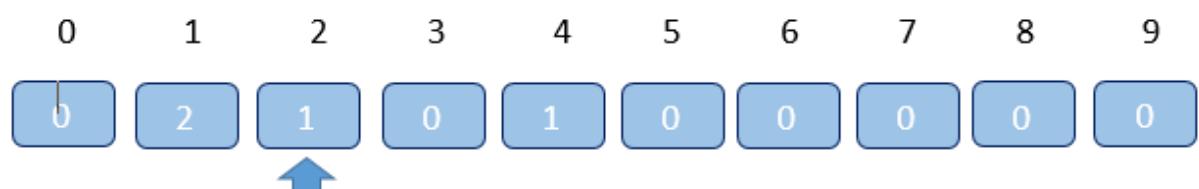


Sekarang elemen ke 4 di myArray yaitu 2



Maka index ke-2 di countArray bertambah 1

Index:

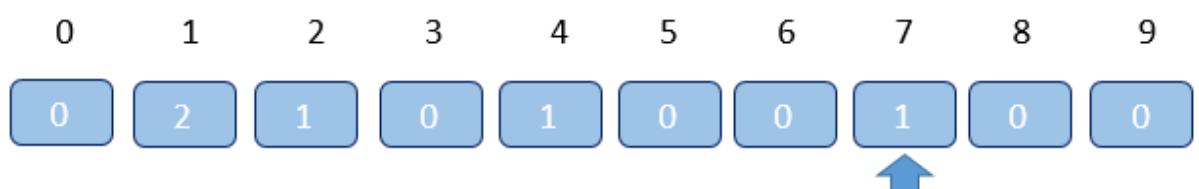


Sekarang elemen 7 di myArray



Maka index 7 di countArray bertambah 1

Index:

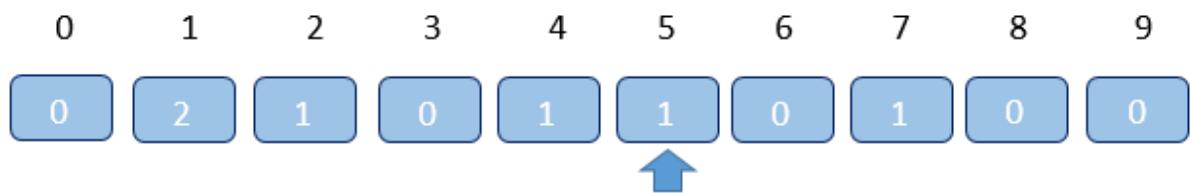


Giliran elemen 5 di myArray



Maka kalian pasti tahu kejadian selanjutnya

Index:

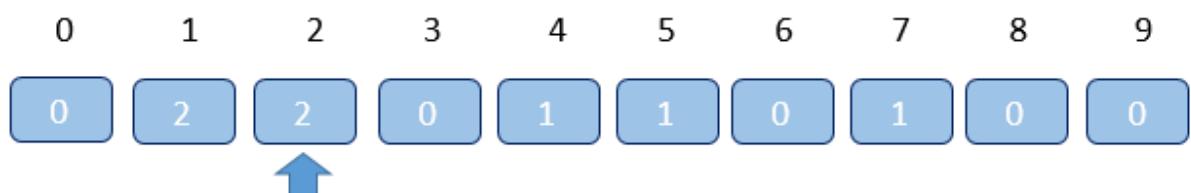


Elemen terakhir di myArray yaitu 2



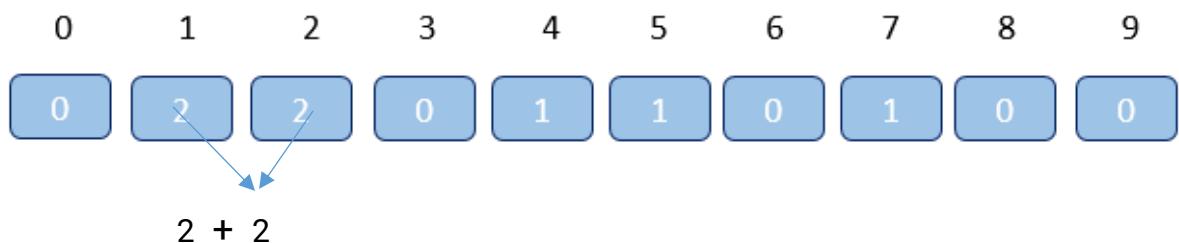
Maka index ke 2 di countArray bertambah satu nilainya

Index:

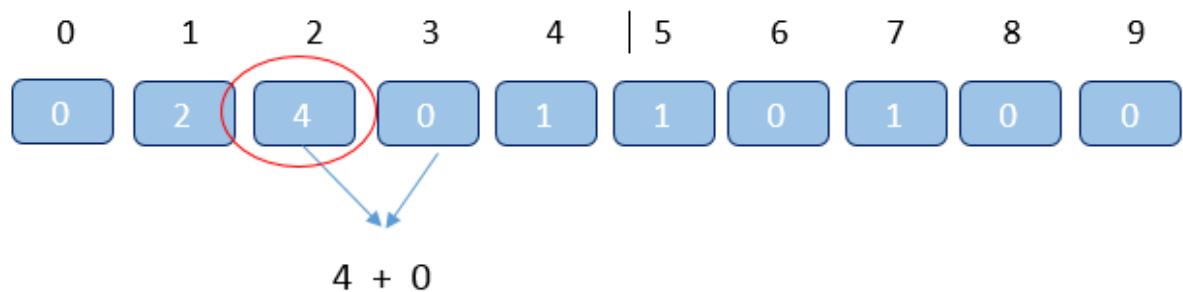


Tahap selanjutnya adalah menjumlahkan index pertama dengan index ke dua dan hasilnya disimpan di index kedua:

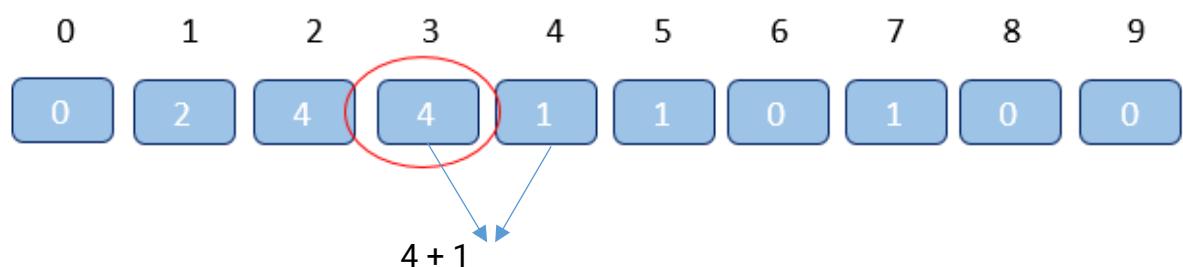
Index:



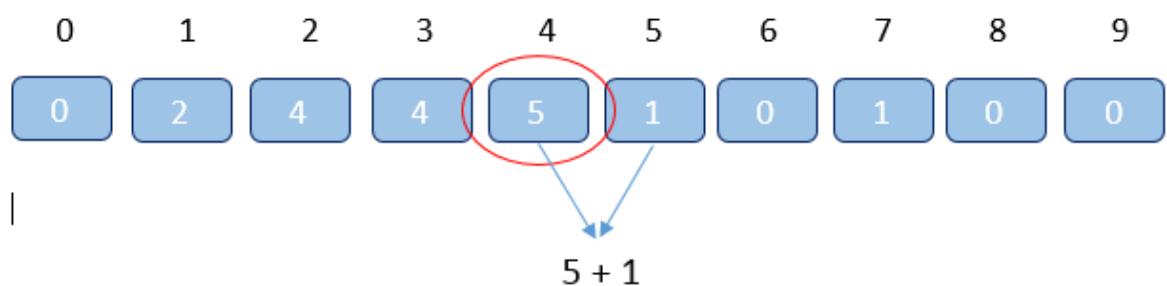
Index:



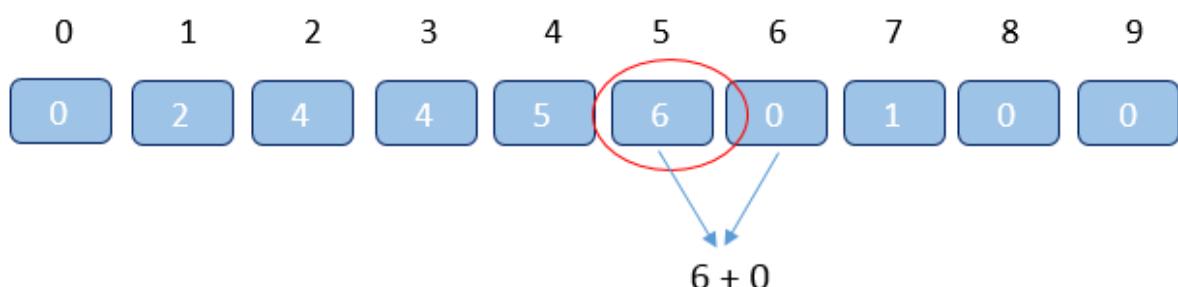
Index:



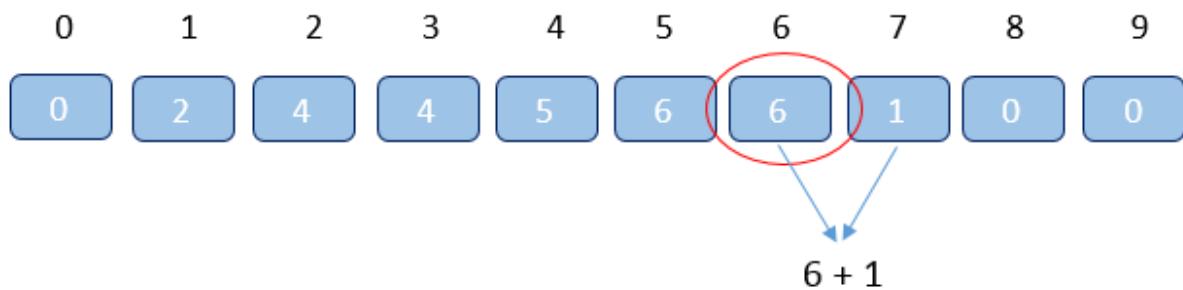
Index:



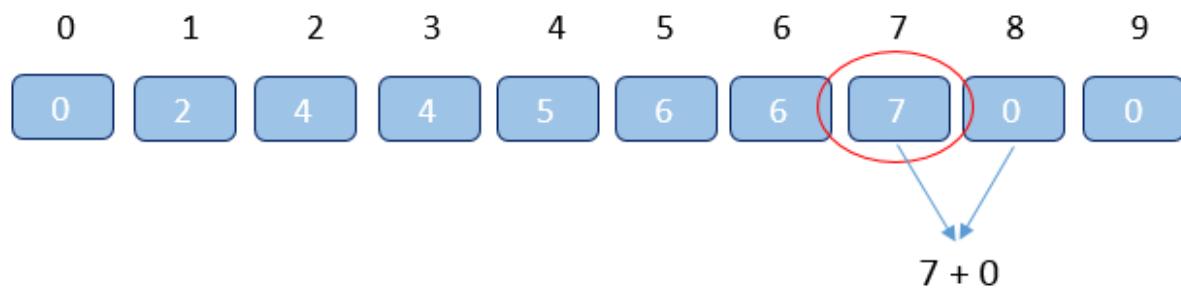
Index:



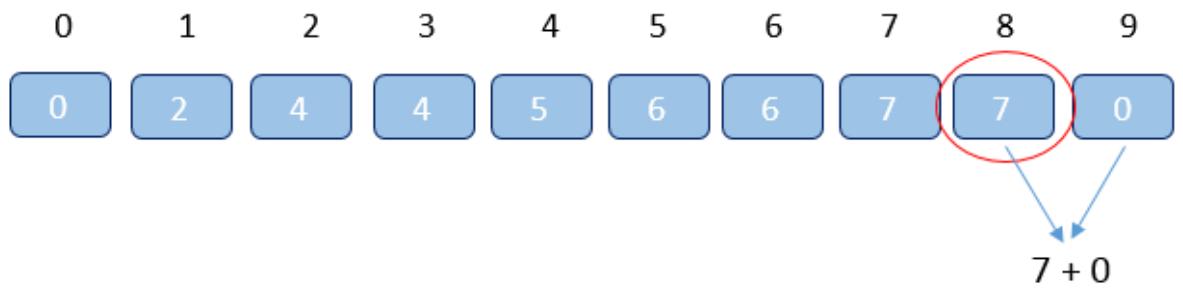
Index:



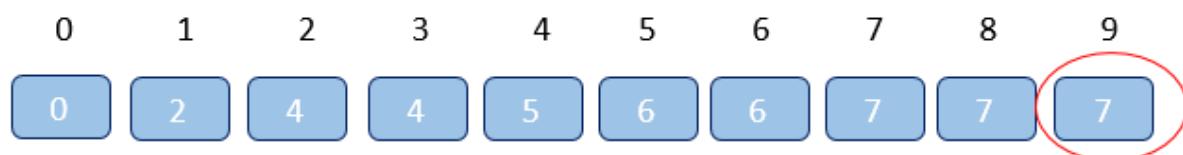
Index:



Index:

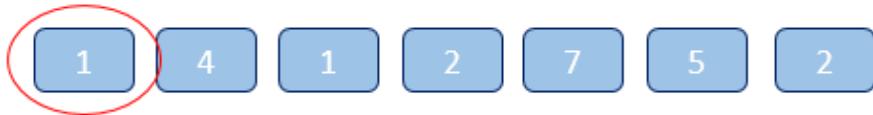


Index :

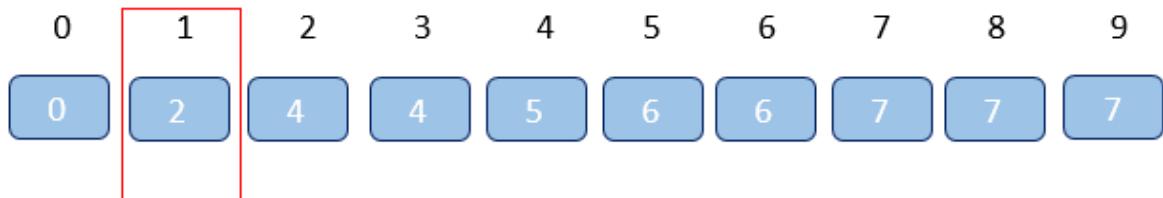


Coba amati tahap selanjutnya di bawah ini:

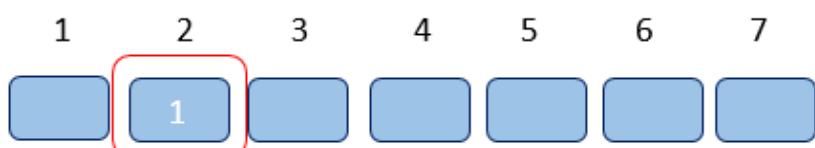
1.



Index :



Index :



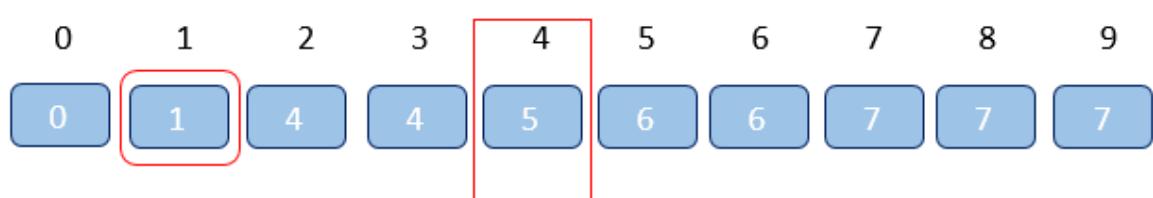
Elemen 1 di myArray menunjukkan index 1 di countArray.

Index 1 di countArray berisi nilai 2, itu berarti index kedua di array yang baru isinya adalah 1.

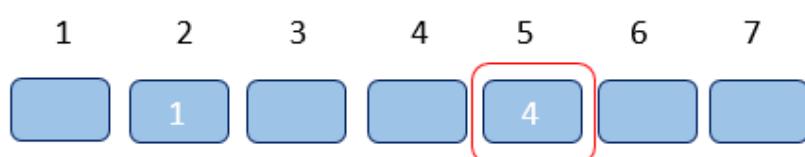
2.



Index:



Index:



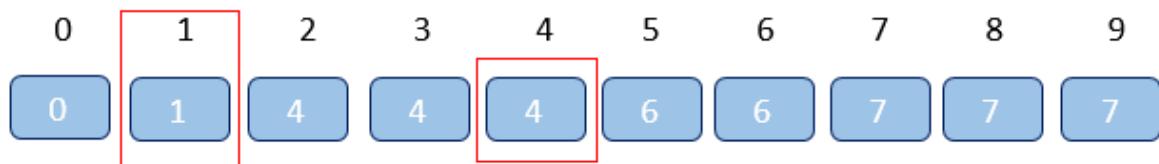
Perhatikan index pertama di countArray nilainya dikurang satu dari dua menjadi satu.

Sekarang elemen 4 di myArray menunjukkan index ke-4 di countArray, yang menunjukkan index ke 5 di array yang baru nilainya adalah 4.

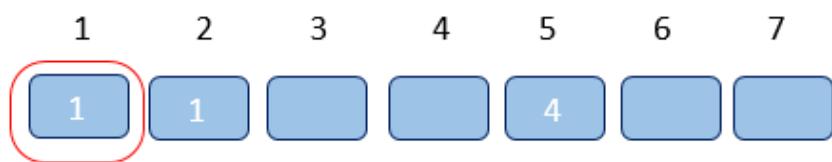
3.



Index:



Index:



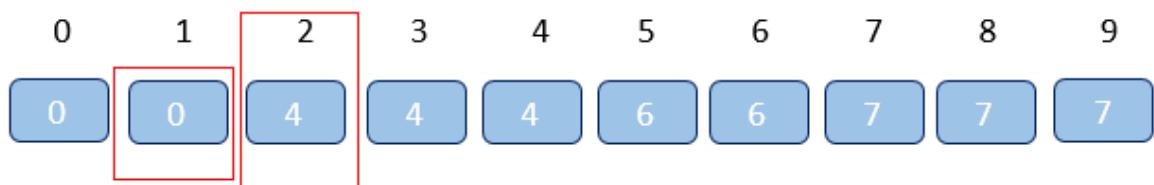
Perhatikan index ke-4 di countArray nilainya dikurang satu dari lima menjadi empat.

Sekarang elemen 1 di myArray menunjukkan index pertama di countArray, yang menunjukkan index ke 1 di array yang baru nilainya adalah 1.

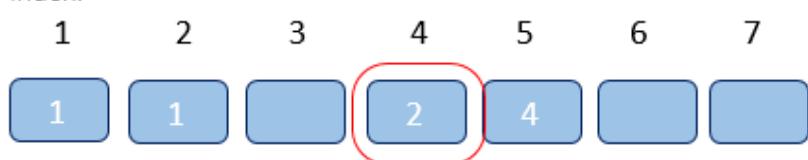
4.



Index:



Index:



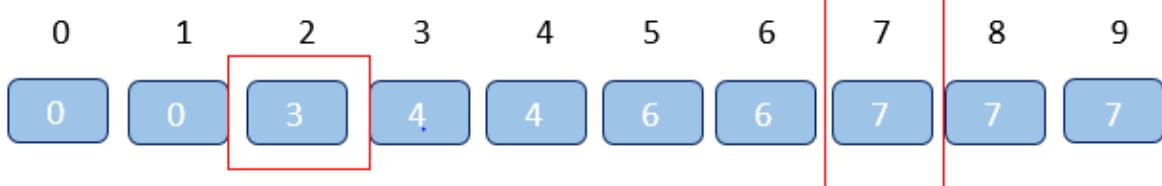
Perhatikan index pertama di countArray nilainya dikurang satu dari satu menjadi nol.

Sekarang elemen 2 di myArray menunjukkan index ke-2 di countArray, yang menunjukkan index ke 4 di array yang baru nilainya adalah 2.

5.



Index:



Index:



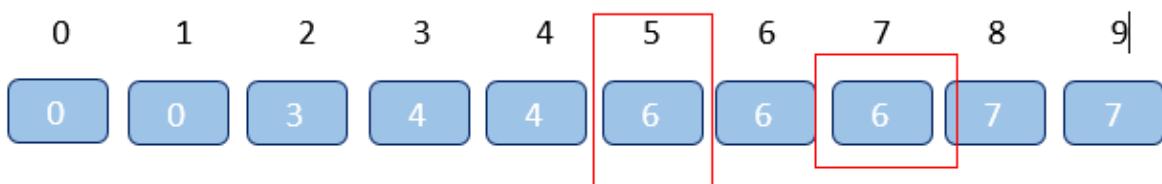
Perhatikan index kedua di countArray nilainya dikurang satu dari empat menjadi tiga.

Sekarang elemen 7 di myArray menunjukkan index ke-7 di countArray, yang menunjukkan index ke 7 di array yang baru nilainya adalah 7.

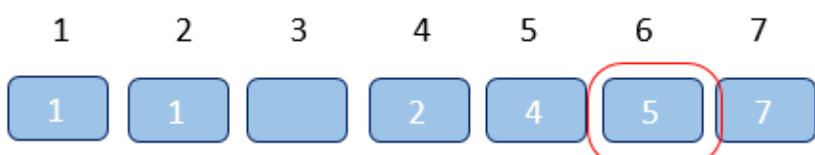
6.



Index:



Index:



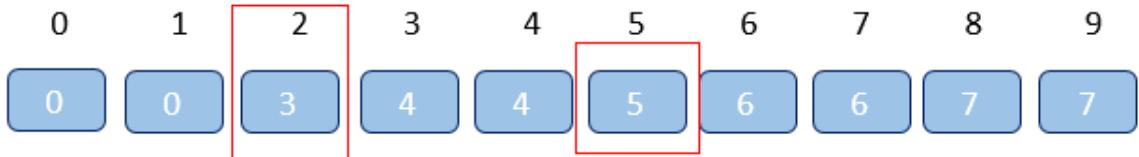
Perhatikan index ke-7 di countArray nilainya dikurang satu dari tujuh menjadi enam.

Sekarang elemen 5 di myArray menunjukkan index ke-5 di countArray, yang menunjukkan index ke 6 di array yang baru nilainya adalah 5.

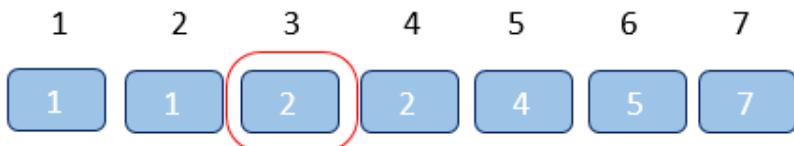
7.



Index:



Index:



Perhatikan index ke-5 di countArray nilainya dikurang satu dari enam menjadi lima.

Sekarang elemen 2 di myArray menunjukkan index ke-2 di countArray, yang menunjukkan index ke 3 di array yang baru nilainya adalah 2.

Keunggulan Algoritma *Counting Sort*

Keunggulan dari algoritma counting sort adalah dapat mengurutkan dengan waktu yang lebih singkat, karena tidak membandingkan dengan elemen lain.

Kelemahan Algoritma *Counting Sort*

Kelemahan algoritma counting sort adalah menggunakan array yang terlalu banyak. Program komputer merupakan instruksi-instruksi logis yang tertulis dalam sebuah source code yang dibaca oleh komputer dan selanjutnya dieksekusi. Salah satu masalah yang dapat diselesaikan oleh komputer adalah pengurutan data. Baik pengurutan data dari yang terbesar ke yang terkecil ataupun sebaliknya.

Implementasi Program

```

#include <stdio.h>
#include <conio.h>

main(){
    int l[20], temp ,j, n=6, idx;
    printf("Masukan 6 Element :\n");

    for(int i=0;i<n;i++){
        printf("Masukan Bilangan : ");scanf("%d",&l[i]);
    }
    printf("\nSebelum di sorting : ");
    for(int i=0; i<n ;i++){
        printf("%d ",l[i]);
    }
    for(int i=0; i<(n-1);i++){
        idx=i;
        for(j=i+1; j<n ; j++){
            if(l[j]<l[idx]){
                idx=j;
            }
        }
        temp=l[i];
        l[i]=l[idx];
        l[idx]=temp;
    }

    printf("\nSetelah di sorting : ");
    for(int i=0; i<n ;i++){
        printf("%d ",l[i]);
    }
    printf("\n");
    getch();
}

```

Output:

```

c:\Pelatihan C\countingSort.exe
Masukan 6 Element :
Masukan Bilangan : 7
Masukan Bilangan : 2
Masukan Bilangan : 1
Masukan Bilangan : 4
Masukan Bilangan : 6
Masukan Bilangan : 4

Sebelum di sorting : 7 2 1 4 6 4
Setelah di sorting : 1 2 4 4 6 7

```

Latihan !

7	5	2	1	5	2	2
---	---	---	---	---	---	---

1. Urutkan secara descending menggunakan descending !

Metode Merge Sort

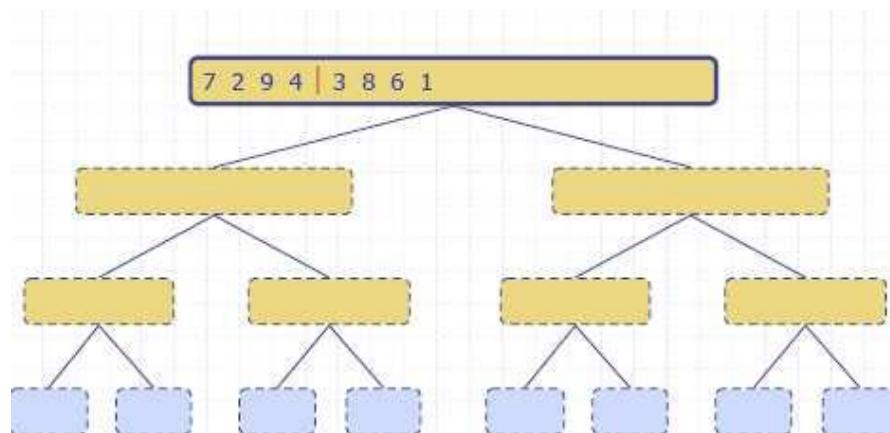
Di antara banyak algoritma pengurutan tersebut terdapat satu kategori algoritma pengurutan yang dalam prosesnya tidak melakukan pembandingan antar data. Yang sering disebut *Non-Comparison Sorting Algorithm*, atau dalam bahasa Indonesia Algoritma Pengurutan-Tanpa-Pembandingan.

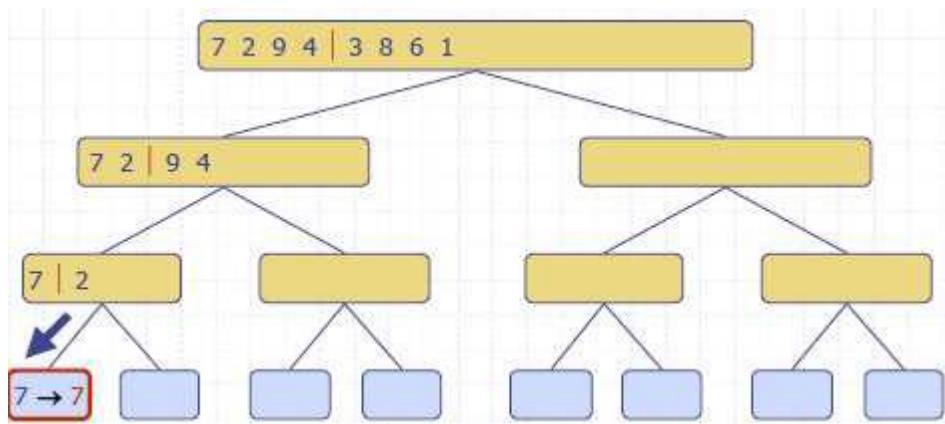
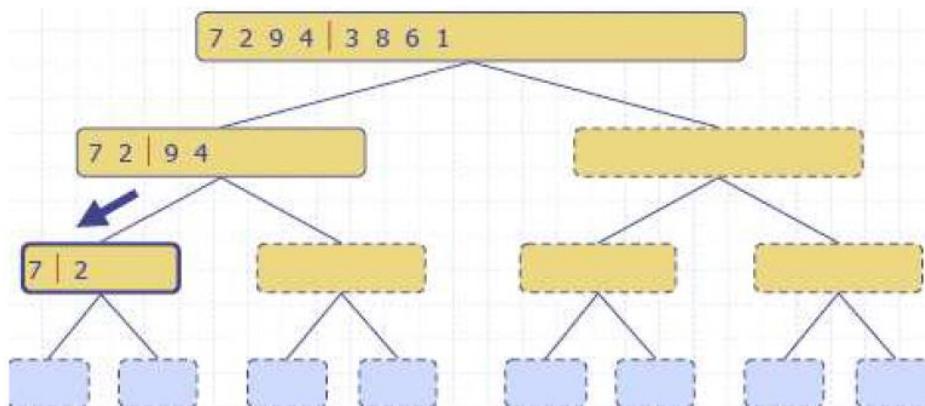
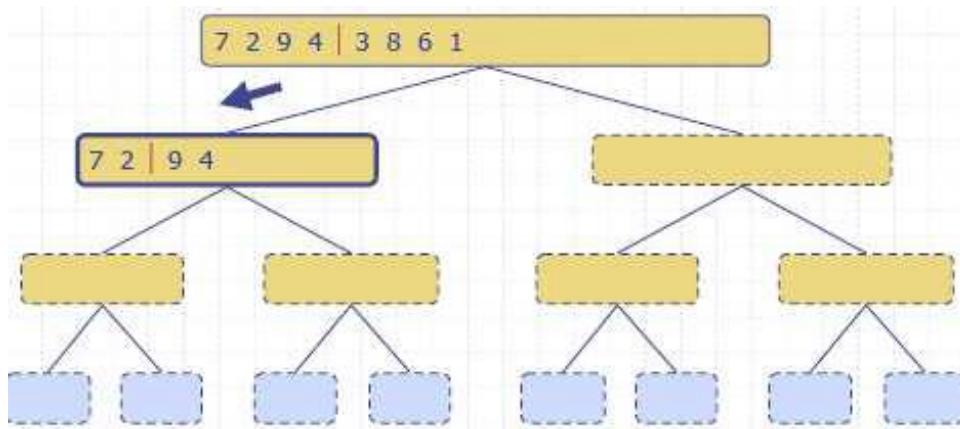
Metode merge sort merupakan metode sorting dengan cara divide and conquer yaitu dengan memecah kemudian menyelesaikan setiap bagian, kemudian menggabungkannya kembali. Pertama data dipecah menjadi 2 bagian di mana bagian pertama merupakan setengah (jika data genap) atau setengah minus satu (jika data ganjil) dari seluruh data, kemudian dilakukan pemecahan kembali untuk masing-masing blok sampai hanya terdiri dari satu data tiap blok.

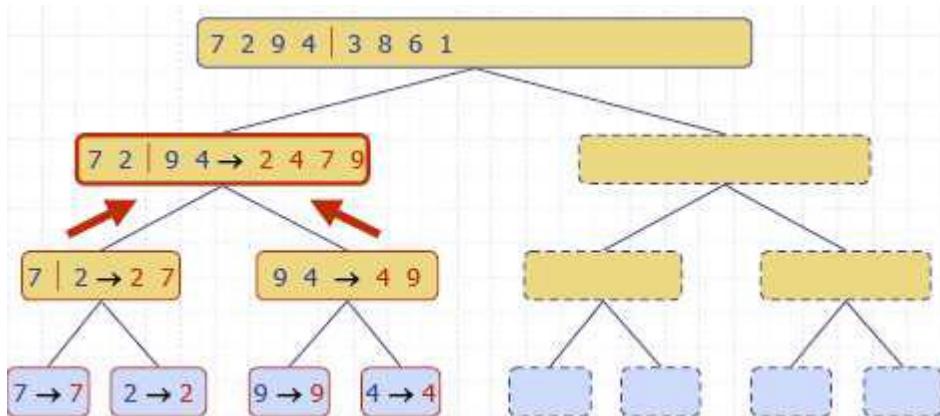
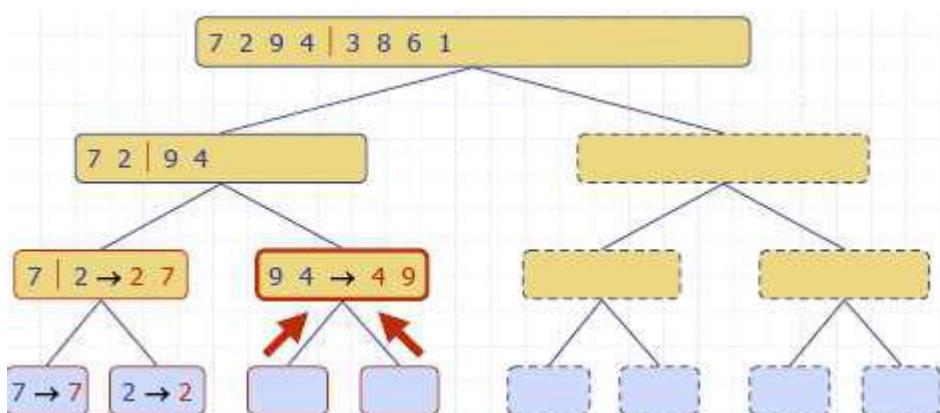
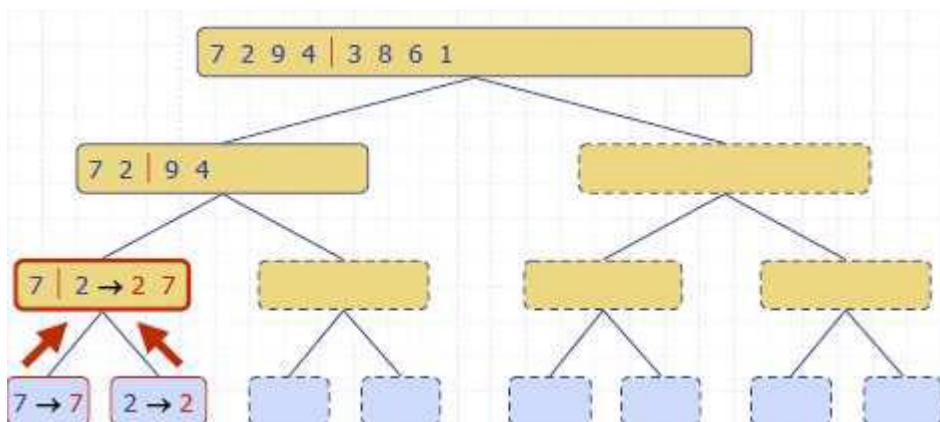
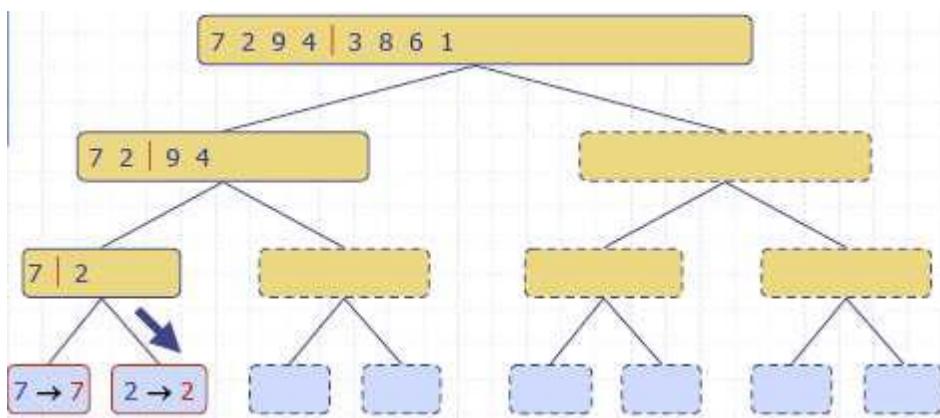
Setelah itu digabungkan kembali dengan membandingkan pada blok yang sama apakah data pertama lebih besar daripada data ke-tengah+1, jika ya maka data ke-tengah+1 dipindah sebagai data pertama, kemudian data ke-pertama sampai ke-tengah digeser menjadi data ke-dua sampai ke-tengah+1, demikian seterusnya sampai menjadi satu blok utuh seperti awalnya. Sehingga metode merge sort merupakan metode yang membutuhkan fungsi rekursi untuk penyelesaiannya.

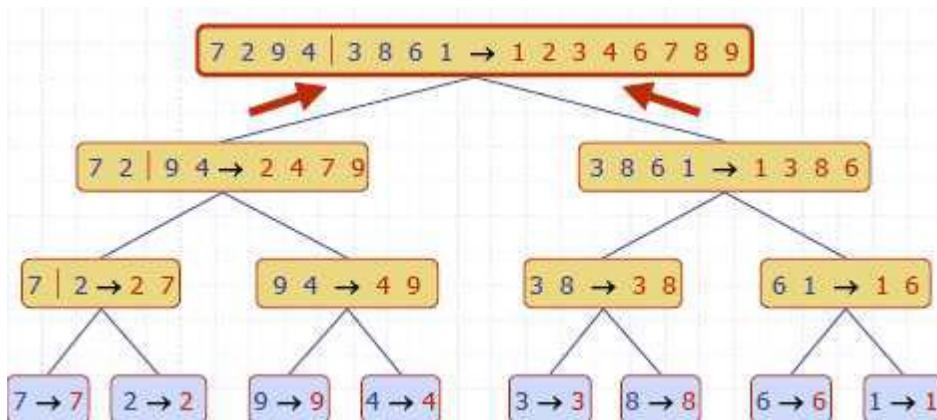
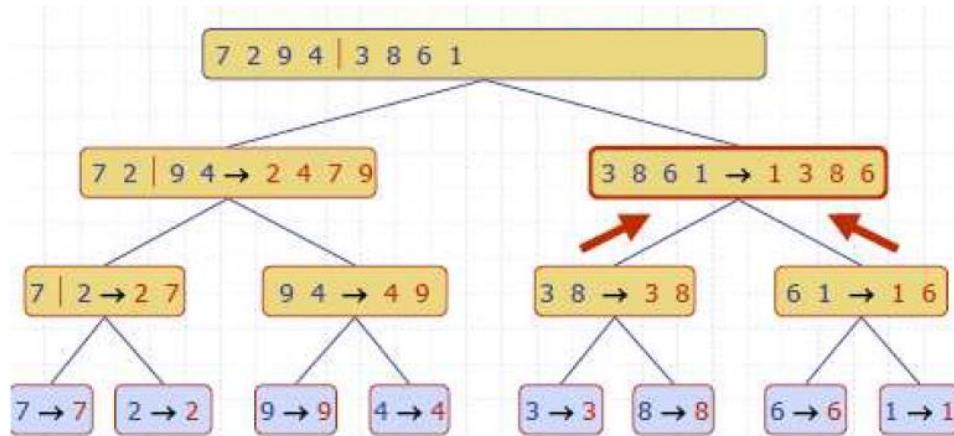
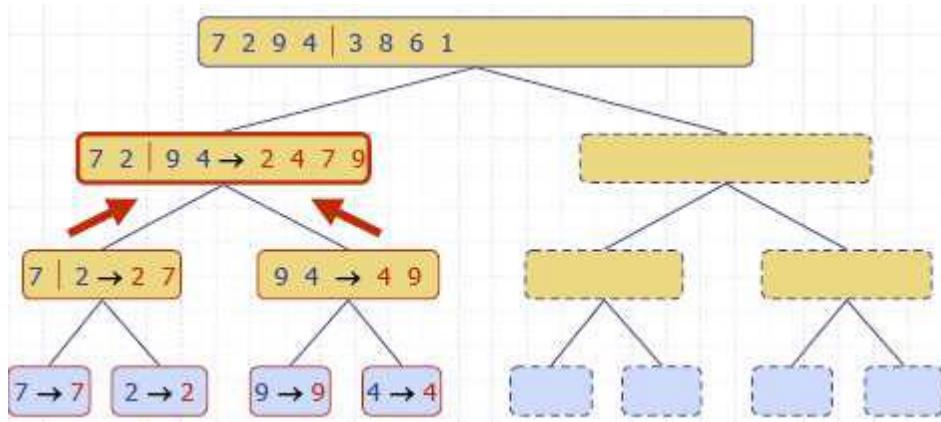
Ilustrasi Algoritma Merge Sort

Amatilah baik-baik ilustrasi Algoritma Merge Sort di bawah ini!









Kelebihan Merge Sort

- Dibanding dengan algoritma lain, merge sort ini termasuk algoritma yang sangat efisien dalam penggunaannya sebab setiap list yang lebih kecil, kemudian digabungkan lagi sehingga tidak perlu melakukan banyak perbandingan.
- Cocok untuk sorting akses datanya lambat misalnya tape drive atau hard disk.
- Cocok untuk sorting data yang biasanya diakses secara sequentially (berurutan), misalnya linked list, tape drive, dan hard disk.

Kekurangan Merge Sort

- terlalu banyak menggunakan ruang pada memori.

- membutuhkan lebih banyak ruang daripada jenis sorting lainnya.

Implementasi Program

```

void merge(int low, int mid, int up);
void mergeSort(int low,int up);

int a[50];

void merge(int low, int mid, int up){
    int h,i,j,k,b[50];
    h = low;
    i = low ;
    j = mid+1;

    while((h<=mid)&&(j<=up)){
        if(a[h] < a[j]){
            b[i] = a[h];
            h++;
        }else{
            b[i] = a[j];
            j++;
        }
        i++;
    }
    if(h>mid){
        for(k=h ; k<=up ; k++){
            b[i]=a[k];
            i++;
        }
    }else{
        for(k=h ; k<=mid ; k++){
            b[i]=a[k];
            i++;
        }
    }

    for(k=low ; k<=up ; k++){
        a[k] = b[k];
    }
}

void mergeSort(int low,int up){
    int mid;
    if(low < up){
        mid = (low+up)/2;
        mergeSort(low,mid);
        mergeSort(mid+1,up);
        merge(low,mid,up);
    }
}

```

```

int main(){
int jumlahBil;

printf("=====Merge Sort=====\\n");
printf("Masukan Jumlah Element array : ");
scanf("%d",&jumlahBil);

for(int i=0;i<jumlahBil;i++){
    printf("bilangan ke -%d :" ,i+1);scanf("%d",&a[i+1]);
}

printf("\\n\\nHasil Merge : ");
mergeSort(1,jumlahBil);

for(int i=0;i<jumlahBil;i++){
    printf("%d",a[i+1]);
}

printf("\\n\\n");
getch();
}

```

Latihan

1.Urutkan data di bawah ini secara ascending dan descending dengan menggunakan metode Merge Sort!

8	1	2	5	12	2	5
---	---	---	---	----	---	---

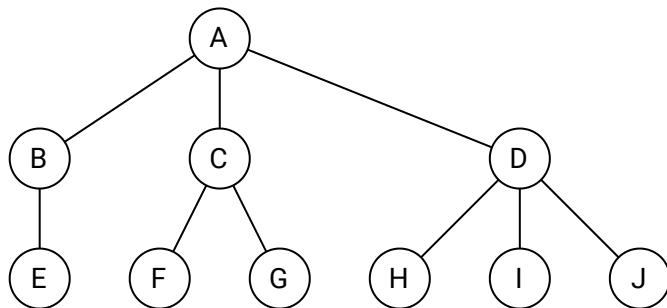
Tree

Daftar isi

- Tree
 - Daftar isi
 - Pengertian
 - Daftar istilah
 - Binary tree
 - Tree traversal
 - In-order
 - Pre-order
 - Post-order
 - Implementasi binary tree dalam C
 - Membuat structure
 - Membuat node baru
 - Membuat fungsi node baru
 - Membuat fungsi tambahkan child
 - Child kiri
 - Child kanan
 - Membuat fungsi cetak tree
 - Binary search tree

Pengertian

Tree (pohon) adalah struktur yang terdiri dari node-node yang saling terhubung yang bersifat hierarki (bertingkat-tingkat dan bercabang-cabang). Tree dalam struktur data tumbuh dari atas ke bawah.



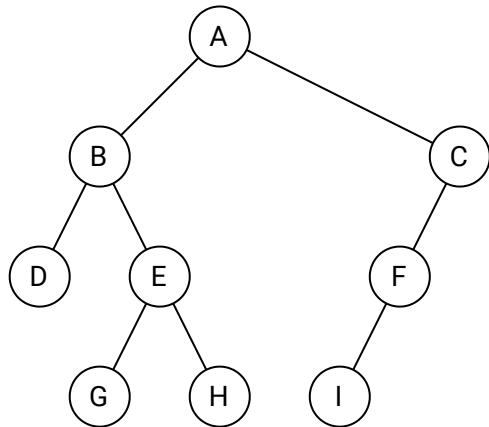
Daftar istilah

Istilah	Terjemahan	Deskripsi
node	simpul	entitas yang berisi data dan pointer ke node child -nya
edge	rusuk	garis penghubung antara 2 node
root	akar	node pertama dalam tree
predecessor	leluhur	node-node sebelum node tertentu
successor	keturunan	node-node setelah node tertentu
ancestor	leluhur sejajar	predecessor pada jalur yang sama

Istilah	Terjemahan	Deskripsi
descendant	keturunan sejajar	successor pada jalur yang sama
parent	induk	ancestor satu level
child	anak	descendant satu level
neighbor	tetangga	node-node yang merupakan parent atau child
sibling	saudara	node-node yang memiliki parent yang sama
subtree	subpohon	node tertentu beserta descendant-nya
internal	internal	node-node yang memiliki child
leaf	daun	node-node yang tidak memiliki child
degree	gelar	jumlah child dari node tertentu
depth	kedalaman	jumlah edge dari root ke node tertentu
height	tinggi	jumlah edge dari descendant terjauh ke node tertentu
height of tree	tinggi pohon	jumlah edge dari node terbawah ke root
size	ukuran	jumlah node dalam tree

Binary tree

Binary tree (pohon biner) adalah tree yang setiap node-nya memiliki maksimal 2 child, tidak bisa lebih.



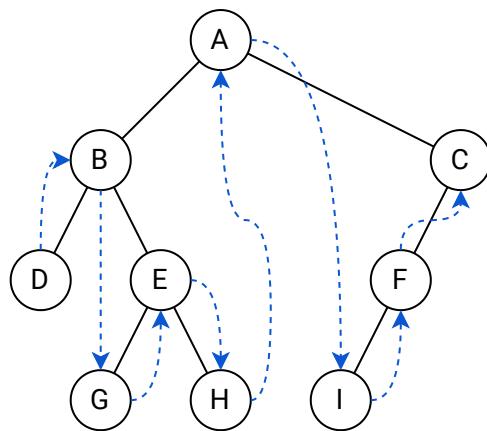
Tree traversal

Tree traversal (lintasan pohon) adalah proses mengunjungi setiap node dalam tree. Tree dapat dilintasi dengan berbagai cara, cara yang paling umum digunakan adalah in-order, pre-order, dan post-order.

In-order

kiri->atas->kanan

Contoh:



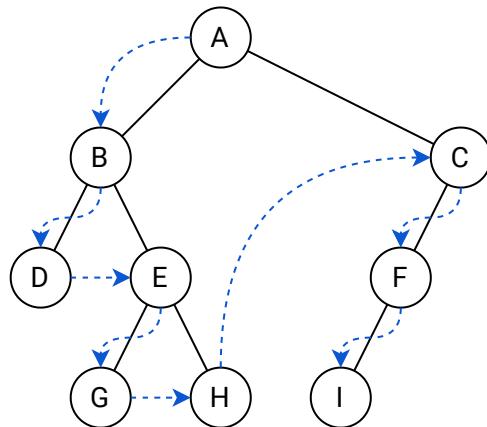
Penjelasan:

- Dimulai dari **successor** kiri terbawah (D)
- Lanjut ke **ancestor** terdekat (B)
- Karena ada **child** lagi (E) maka lanjut ke **successor** kiri terbawah dari **child** itu (G)
- Lanjut ke **ancestor** terdekat (E)
- Karena ada **child** lagi (H) maka lanjut ke **successor** kiri terbawah dari **child** itu (H itu sendiri)
- Lanjut ke **ancestor** terdekat yang belum dikunjungi (A)
- Karena ada **child** lagi (C) maka lanjut ke **successor** kiri terbawah dari **child** itu (I)
- Lanjut ke **ancestor** terdekat (F)
- Karena tidak ada **child** maka lanjut ke **ancestor** terdekat (C)

Pre-order

atas->kiri->kanan

Contoh:



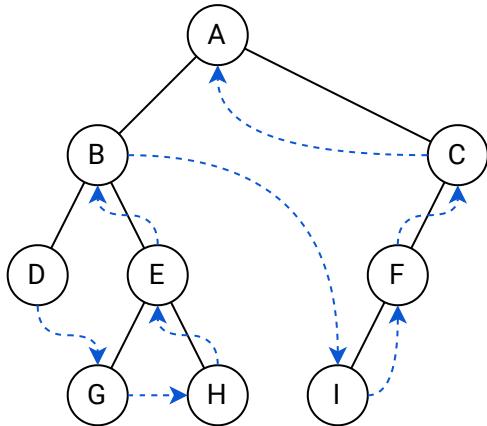
Penjelasan:

- Dimulai dari **root** (A)
- Karena ada **child** (B) maka lanjut ke **child** itu (B)
- Karena ada **child** (D) maka lanjut ke **child** itu (D)
- Karena tidak ada **child** maka lanjut ke **sibling** (E)
- Karena ada **child** (G) maka lanjut ke **child** itu (G)
- Karena tidak ada **child** maka lanjut ke **sibling** (H)
- Karena tidak ada **child** maupun **sibling**, maka lanjut ke **child** dari **ancestor** yang **child** itu belum dikunjungi
- Karena ada **child** (F) maka lanjut ke **child** itu (F)
- Karena ada **child** (I) maka lanjut ke **child** itu (I)

Post-order

kiri->kanan->atas

Contoh:



Penjelasan:

- Dimulai dari **successor** kiri terbawah (D)
- Karena ada **sibling** (E) maka lanjut ke **successor** kiri terbawah dari **sibling** itu (G)
- Karena ada **sibling** (H) maka lanjut ke **successor** kiri terbawah dari **sibling** itu (H itu sendiri)
- Karena tidak ada **sibling** lagi maka lanjut ke **parent** (E)
- Karena tidak ada **sibling** lagi maka lanjut ke **parent** (B)
- Karena ada **sibling** (C) maka lanjut ke **successor** kiri terbawah dari **sibling** itu (I)
- Karena tidak ada **sibling** lagi maka lanjut ke **parent** (F)
- Karena tidak ada **sibling** lagi maka lanjut ke **parent** (C)
- Karena tidak ada **sibling** lagi maka lanjut ke **parent** (A)

Implementasi binary tree dalam C

Membuat structure

Seperti linked-list, pertama kita harus membuat structure menggunakan **struct** bernama **Node** yang berisi data dan pointer. Pointer itu berfungsi untuk terhubung ke **child**. Karena setiap node dalam binary tree hanya dapat memiliki maksimal 2 **child**, maka **child** pertama disebut **left** (kiri) dan **child** kedua disebut **right** (kanan).

```

struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
};
  
```

Membuat node baru

```

void main()
{
    struct Node *node = malloc(sizeof(struct Node));
    node->data = 7;
  
```

```
node->left = NULL;
node->right = NULL;
}
```

Kode di atas akan membuat node baru dengan data bilangan 7 dan tidak memiliki **child**.

Membuat fungsi node baru

Tentunya kita tidak hanya membuat 1 node, tapi banyak node. Agar tidak perlu mengulang-ulang menulis kode untuk membuat node, maka kita buat fungsi yang akan mengembalikan node baru.

```
struct Node *new_node(int data)
{
    struct Node *node = malloc(sizeof(struct Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
```

Setelah kita membuat fungsi di atas, untuk membuat node baru kita hanya perlu memanggil fungsi itu dengan data sebagai argumennya.

Contoh:

```
void main()
{
    // membuat node baru
    struct Node *a = new_node(7);

    // mencetak data node
    printf("%d", a->data);
}
```

Membuat fungsi tambahkan child

Child kiri

```
struct Node *insert_left(struct Node *parent, int data)
{
    parent->left = new_node(data);
    return parent->left;
}
```

Child kanan

```
struct Node *insert_right(struct Node *parent, int data)
{
    parent->right = new_node(data);
    return parent->right;
}
```

Membuat fungsi cetak tree

```
void print(struct Node *node, int level, bool last)
{
    for (int i = 0; i < level; i++)
    {
        if (i == level - 1)
        {
            printf("+-");
        }
        else if (i == level - 2 && !last)
        {
            printf("| ");
        }
        else
        {
            printf("  ");
        }
    }
    printf("%d\n", node->data);
    level++;
    if (node->left)
    {
        print(node->left, level, false);
    }
    if (node->right)
    {
        print(node->right, level, true);
    }
}
```

Binary search tree

Binary search tree (pohon pencarian biner) adalah struktur data binary tree berbasis node yang memiliki properti berikut:

- Subtree kiri dari sebuah node hanya berisi node-node dengan nilai yang lebih kecil.
- Subtree kanan dari sebuah node hanya berisi node-node dengan nilai yang lebih besar.
- Subtree kiri dan kanan masing-masing juga harus berupa binary search tree.

Graph

Daftar isi

- [Graph](#)
 - [Daftar isi](#)
 - [Pengertian](#)

Pengertian

Graph adalah sekelompok simpul-simpul (nodes/vertices) V , dan sekelompok sisi (edges) E yang menghubungkan sepasang simpul. Bayangkan simpul-simpul tersebut sebagai lokasi-lokasi, maka himpunan dari simpul-simpul tersebut adalah himpunan lokasi-lokasi yang ada. Dengan analogi ini, maka sisi merepresentasikan jalan yang menghubungkan pasangan lokasi-lokasi tersebut.

Coming soon...