Struktur Data dan Algoritma

Anita Sindar RMS, ST, MTI

BAB I
PENDAHULUAN

A. Pengantar

Suatu sistem aplikasi pengolahan data dengan representasi data berbentuk matrik sparse. Bagaimana struktur datanya? Suatu sistem mengintegrasikan beberapa sistem lainnya. Bagaimana struktur datanya? Isu struktur data semakin penting pada saat: Ukuran data sangat besar, data sangat komplek, kebutuhan akses data sangat cepat, adanya keterbatasan sumberdaya. Struktur data lebih kongkret, merupakan teknik atau strategi untuk mengimplementasikan sebuah ADT (ADT lebih merupakan deskripsi logika) Struktur data merupakan cara membentuk, mengkonstruksi, mengaransemen, mengkomposisikan ataupun mengorganisasikan data (ADT) ADT: stack, queue, priority queue, dictionary, sequence, set Struktur Data: array, linked list, hash table (open, closed, circular hashing), trees (binary search trees, heaps, AVL trees, 2-3 trees, tries, red/black trees, B-trees).

B. Deskripsi

Mata kuliah ini memberi pemahaman mengenai Konsep Struktur Data, Konsep Algoritma Pemograman, Defenisi Algoritma, Tipe Data, Pseudo - Code , Sintak dalam Pemograman C++, Standard I/O pemograman, Perintah Masukan, Perintah Keluaran, Mengakses Element Structure, Array dan Structure, Nested Structure, Operator '*' dan '&', Array Dimensi Satu, Array Dimensi Dua, Array Multi Dimensi, Structure String, Prototype Konversi String, Passing Variable (by reference / by value), Return Value/ Non Return Value, Insertion Sort, Bubble Sort, Selection Sort, Squencial Search Un-Order List, Squencial Search Order, Binary Search, Linked List, Pointer-Based Linked List, Circular linked list, Operasi dan aplikasi Stack, Operasi dan Aplikasi Queue dan Deque, Algoritma Push dan Pop, FIFO, Terminologi Tree, Karakteristik Tree, Terminologi Graph, Traversal Graph.

C. Tujuan Instruksional Umum

Memberikan pengetahuan tentang konsep struktur data dan algoritma, Sintak dalam Pemograman C++, Structure, Pointer, Array dan String, Function, Sorting, Searching, Linked List, Stack, Quee, Tree, Graph.

D. Capaian Pembelajaran

- Mahasiswa memahami dan menerapkan konsep pengorganisasian kumpulan data dan algoritma pemrograman struktur data dalam pemrograman.
- Mahasiswa memahami dan menjelaskan definisi struktur data dan algoritma pemograman.
- Mahasiswa mengenal dan memahami Pemograman C++.
- Mampu memahami, menjelaskan dan menerapkan definisi struktur, array dan pointer.
- Mampu memahami, menjelaskan dan menerapkan sorting, branching, looping.
- Mampu memahami, menjelaskan dan menerapkan function pada pemrograman.
- Mampu memahami, menjelaskan dan menerapkan definisi stack dan queque.
- Mampu memahami, menjelaskan dan menerapkan definisi single double linked list.
- Mampu memahami, menjelaskan dan menerapkan definisi tree dan graph.

E. Proses Pembelajaran

Teori Dan Praktek, Tatap Muka, Infokus, White Board, Penghapus, Laptop

F. Penilaian
Kehadiran = 10%
Tugas/Quis = 20%
UTS = 30%
UAS = 40%

MODUL 1
KONSEP STRUKTUR DATA DAN ALGORITMA

A. Tujuan

Mahasiswa mampu memahami konsep struktur data, defenisi struktur data, defenisi algoritma, mengenal C++, paradigm pemograman, implementasi pemograman, kerangka program C++, komentar, identifier.

B. Uraian Materi

Materi 1 : Konsep Struktur Data

Struktur data adalah cara menyimpan atau merepresentasikan data di dalam komputer agar bisa dipakai. Struktur data adalah sebuah skema organisasi, seperti struktur dan array, yang diterapkan pada data sehingga data dapat diinterprestasikan, sehingga operasi-operasi spesifik dapat dilaksanakan pada data tersebut. Data adalah representasikan dari fakta dunia nyata.

Fakta atau keterangan tentang kenyataan yang disimpan, direkam atadirepresentasikan dalam bentuk tulisan suara, gambar, atau symbol. Suatu struktur data mempunyai tiga bagian utama yaitu :

- Himpunan struktur dari tempat penyimpanan atau storage.
  Merupakan koleksi dari variable dan hubungan antara satu variable dengan variable yang lain.
- Himpunan dari fungsi-fungsi dasar.
  Dapat digunakan pada struktur tempat penyimpanan yang ada dan dapat digunakan pada setiap bagian dari program
- Himpunan dan algoritma digunakan untuk pengubahan dari struktur tempat penyimpanan.

Materi 2 : Defenisi Algoritma

- Algoritma adalah urutan aksi-aksi yang dinyatakan dengan jelas dan tidak rancu untuk memecahkan suatu masalah dalam rentang waktu tertentu. Setiap aksi harus dapat dikerjakan dan mempunyai efek tertentu.
- Algoritma dapat dituliskan dengan banyak cara, mulai dari menggunakan bahasa alami yang digunakan sehari-hari, simbol grafik bagan alir, sampai menggunakan bahasa pemograman seperti bahasa C atau C++.
- Algoritma adalah barisan langkah-langkah perhitungan dasar yang mengubah masukan (dari beberapa fungsi matematika) menjadi keluaran.
  Contoh :
  Perkalian
  Input : integer positif a, b
  Output : a x b
  Algoritma perkalian :
  Contoh kasus : a = 365, b = 24
  Metode 1 : 365 * 24 = 365 + (365 * 23)
  = 730 + (365 * 22)
  …..
  = 8760 + (365 * 0)
  = 8760
- Algoritma adalah urutan aksi-aksi yang dinyatakan dengan jelas dan tidak rancu untuk memecahkan suatu masalah dalam rentang waktu tertentu.
- Algoritma merupakan logika, metode dan tahapan (urutan) sistematis yang digunakan untuk memecahkan suatu permasalahan.
- Algoritma dapat dituliskan dengan banyak cara, mulai dari menggunakan bahasa alami yang digunakan sehari-hari, symbol grafik bagan alir (flowchart), sampai menggunakan bahasa pemrograman seperti bahasa C atau C++.

- Program adalah kompulan instruksi komputer, sedangkan metode dan tahapan sistematis dalam program adalah algoritma. Program ini ditulis dengan menggunakan bahasa pemrograman, disimpulkan program adalah suatu implementasi dari bahasa pemrograman.

• Data

Defenisi : fakta atau kenyataan sebuah objek

Pengertian : suatu nilai yang mengandung arti memiliki tipe konstanta atau variabel

• Algoritma

Algoritma adalah urutan aksi-aksi yang dinyatakan dengan jelas dan tidak rancu untuk memecahkan suatu masalah dalam rentang waktu tertentu.

Algoritma dapat dituliskan dengan banyak cara, mulai dari menggunakan bahasa alami yang digunakan sehari-hari, simbol grafik bagan alir, sampai menggunakan bahasa
pemograman seperti bahasa C atau C++.

• Variabel

Variabel adalah suatu pengenal (identifier) yang digunakan untuk mewakili suatu nilai tertentu di dalam proses program. Berbeda dengan konstanta yang nilainya selalu tetap, nilai dari suatu variable bisa diubah-ubah sesuai kebutuhan. Untuk memperoleh nilai dari suatu variable digunakan pernyataan penugasan (assignment statement), yang mempunyai sintaks sebagai berikut :

variable = ekspresi ;

• Tipe data

Tipe data dapat dikelompokkan menjadi atas dua macam :

1. Tipe sederhana untuk menyimpan satu nilai dalam satu variabel
2. Tipe data terstruktur

• Definisi Program/Pemrograman

Adalah kumpulan instruksi-instruksi tersendiri yang biasanya disebut source code yang dibuat oleh programmer (pembuat program)

• Paradigma Pemrograman

1. Pemrograman Prosedural
   - Berdasarkan urutan-urutan, sekuensial
   - Program adalah suatu rangkaian prosedur untuk memanipulasi data. Prosedur merupakan kumpulan instruksi yang dikerjakan secara berurutan.
2. Pemrograman Fungsional
   - Berdasarkan teori fungsi matematika
   - Fungsi merupakan dasar utama program.
3. Pemrograman Terstruktur
   - Secara berurutan dan terstrukrtur.
   - Program dapat dibagai-bagi menjadi prosedur dan fungsi.

   Contoh: PASCAL dan C
4. Pemrograman Modular
   - Pemrograman ini membentuk banyak modul.
   - Modul merupakan kumpulan dari prosedur dan fungsi yang berdiri sendiri. Sebuah program dapat merupakan kumpulan modul-modul.

Contoh: MODULA-2 atau ADA

5. Pemrograman Berorientasi Obyek
   - Pemrograman berdasarkan prinsip obyek, dimana obyek memiliki
   - data/variabel/property dan method/event/prosedur yang dapat dimanipulasi.

   Contoh: C++, Object Pascal, dan Java.
6. Pemrograman Berorientasi Fungsi

   Pemrograman ini berfokus pada suatu fungsi tertentu saja.

Contoh: SQL (Structured Query Language), HTML, XML dan lain-lain.

7. Pemrograman Deklaratif

Pemrograman ini mendeskripsikan suatu masalah dengan pernyataan daripada memecahkan masalah dengan implementasi algoritma.

Contoh: PROLOG

• Kriteria Algoritma Menurut Donald E. Knuth

1. Input: algoritma dapat memiliki nol atau lebih inputan dari luar.
2. Output: algoritma harus memiliki minimal satu buah output keluaran.
3. Definiteness (pasti): algoritma memiliki instruksi-instruksi yang jelas dan tidak ambigu.
4. Finiteness (ada batas): algoritma harus memiliki titik berhenti (stopping role).
5. Effectiveness (tepat dan efisien): algoritma sebisa mungkin harus dapat dilaksanakan dan efektif. Contoh instruksi yang tidak efektif adalah: A = A + 0 atau A = A * 1

• Jenis Proses Algoritma:

1. Sequence Process: instruksi dikerjakan secara sekuensial, berurutan.
2. Selection Process: instruksi dikerjakan jika memenuhi kriteria tertentu
3. Iteration Process: instruksi dikerjakan selama memenuhi suatu kondisi tertentu.
4. Concurrent Process: beberapa instruksi dikerjakan secara bersama.

Contoh Algoritma

Algoritma menghitung luas persegi panjang:

1. Masukkan panjang (P)
2. Masukkan lebar (L)
3. L ← P * L
4. Tulis L

Definisi Pseudo-code

Kode atau tanda yang menyerupai (pseudo) atau merupakan pejelasan cara menyelesaikan suatu masalah. Pseudo-code sering digunakan oleh manusia untuk menuliskan algoritma. Problem: mencari bilangan terbesar dari dua bilangan yang diinputkan

Contoh Pseudo-code:

1. Masukkan bilangan pertama
2. Masukkan bilangan kedua
3. Jika bilangan pertama > bilangan kedua maka kerjakan langkah 4, jika tidak, kerjakan Langkah 5.
4. Tampilkan bilangan pertama
5. Tampilkan bilangan kedua

Contoh Algoritma

1. Masukkan bilangan pertama (a)
2. Masukkan bilangan kedua (b)
3. if a > b then kerjakan langkah 4
4. print a
5. print b

1. Mendefinisikan masalah

Tentukan masalahnya, apa saja yang harus dipecahkan dengan menggunakan komputer, dan apa inputan serta outputnya.

2. Menemukan solusi

langkah berikutnya adalah menentukan solusi. Jika masalah terlalu kompleks, maka ada baiknya masalah tersebut dipecah menjadi modul-modul kecil agar lebih mudah diselesaikan. Contohnya masalah invers matriks, maka kita dapat membagi menjadi beberapa

modul:

- meminta masukkan berupa matriks bujur sangkar
- mencari invers matriks
- menampilkan hasil kepada pengguna

3. Memilih algoritma
Pilihlah algoritma yang benar-benar sesuai dan efisien untuk permasalahan tersebut
4. Menulis program
Pilihlah bahasa yang mudah dipelajari, mudah digunakan, dan lebih baik lagi jika sudah dikuasai, memiliki tingkat kompatibilitas tinggi dengan perangkat keras dan platform lainnya.
5. Menguji program
silahkan uji program tersebut dengan segala macam kemungkinan yang ada, termasuk error-handlingnya sehingga program tersebut akan benar-benar handal dan layak digunakan.
6. Menulis dokumentasi
Caranya adalah dengan menuliskan komentar-komentar kecil tentang apa maksud kode tersebut, untuk apa, variabel apa saja yang digunakan, untuk apa, dan parameter-parameter yang ada pada suatu prosedur dan fungsi.
7. Merawat program
Program yang sudah jadi perlu dirawat untuk mencegah munculnya bug yang sebelumnya tidak terdeteksi. Atau mungkin juga pengguna membutuhkan fasilitas baru yang dulu tidak ada.

Materi 4 : Implementasi Pemograman

- C++ merupakan perluasan bahasa C dengan tambahan fasilitas kelas (Class).
- Program C++ berupa sekumpulan fungsi. Bahkan program utama juga berbentuk fungsi, yaitu fungsi main().
- Kode C++ bersifat *case sensitive*, artinya membedakan antara huruf kapital dengan huruf kecil.
- Bahasa C dan C++ merupakan bahasa yang sangat populer dalam dunia pengembangan perangkat lunak. Kedua bahasa ini digolongkan ke dalam bahasa tingkat menengah (*middle level language*).
- Keistimewaan dari bahasa C++ adalah karena bahasa ini mendukung pemrograman berarah objek atau yang lebih sering dikenal dengan istilah *Object Oriented Programming* (OOP).

Object Oriented Programming (OOP)

- Mempermudah programmer menulis program.
- Mempercepat proses pembuatan program.
- Mempermudah pemeliharaan program.

Kerangka Program C++

```
#include <iostream.h>
//Protopype fungsi
tipe_data nama_fungsi(parameter1,parameter2,..);
//Fungsi utama
void main()
{
statemen_yang_akan_dilakukan;
...
return 0;
}
//Implementasi fungsi
tipe_data nama_fungsi(parameter1,parameter2,...)
{
statemen_yang_akan_dilakukan;
...
        }
#include<iostream.h>
int main()
{
cout <<"Selamat menggunakan C++";
return 0;
        }
```

Keterangan:

- #include adalah sebuah prosesor pengarah yang mengatakan kepada kompiler untuk meletakan kode dari header file iostream.h kedalam program. Fungsi cout memerlukan file iostream.h.
- Main adalah nama judul fungsi.
- Tanda () digunakan untuk mengapit argumen fungsi yaitu nilai yang akan dilewatkan ke fungsi.

- Tanda { pada fungsi main() menyatakan awal eksekusi program. Adapun } pada fungsi main() menyatakan akhir eksekusi program.
- Pemakaian fungsi cout dipakai untuk menampilkan text.
- Memakai tanda atau symbol <<, yang diketahui sebagai operator pemasukan (insertion operators). Tanda tersebut mengatakan kepada kompiler agar segera menghasilkan output sesuai dengan input.
- *"Selamat menggunakan C++"* adalah suatu pernyataan yang diapit oleh tanda petik ganda.
- Setiap pernyataan harus diakhiri dengan tanda **titik koma (;)**.
- return 0 maksudnya pada baris ini juga ada kode yang memerintahkan fungsi main kembali ke 0.
- Komentar merupakan bagian yang penting dalam program.
- Komentar tidak akan mempengaruhi terhadap jalannya program karena komentar tidak ikut dieksekusi pada saat proses kompilasi.

Fungsi komentar antara lain:

1. Menjelaskan tujuan / fungsi program.
2. Memudahkan saat program dibuat atau direvisi.
3. Menjelaskan keterangan-keterangan lain tentang kegunaan sejumlah pernyataan dalam program.

- Menggunakan tanda //

  Pada C++ suatu komentar diawali dengan dua tanda garis miring (//). Semua tulisan yang terletak sesudah tanda // hingga akhir baris dengan sendirinya akan diperlakukan sebagai keterangan. Tanda ini digunakan untuk menuliskan komentar yang banyaknya hanya satu baris.
- Menggunakan tanda /*.....*/

  Tanda ini dapat digunakan untuk menuliskan komentar yang banyaknya satu baris atau lebih. Bentuk ini bermanfaat untuk mengabaikan sejumlah pernyataan yang telah dibuat oleh pemrograman karena suatu alasan misalnya sedang melacak kesalahan.

***Identifier* (pengenalan)** adalah suatu nama yang biasa dipakai dalam pemrograman untuk menyatakan variabel, konstanta bernama, tipe data, fungsi, label, objek. Indentifikasi dilakukan untuk mempermudah proses penanganan data atau nilai.

Contoh identifier: int bilangan_bulat, long X2, int A[5], const int MAX=5, int A=10, B=15, C=25;

C. Rangkuman

- Pemakaian struktur data yang tepat di dalam proses pemrograman akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana.
- Secara umum struktur data terdiri dari beberapa bagian seperti himpunan nilai-nilai data dan sejumlah operasi dasar yang bekerja pada data tersebut menurut suatu algoritma tertentu.

D. Tugas

1. Membuat resume dari materi yang telah diterangkan.
2. Tanya jawab dan diberikan latihan soal terhadap individu.

E. Test

1. Jelaskan konsep struktur data.
2. Jelaskan defenisi algoritma.
3. Jelaskan implementasi struktur data dan algoritma.
4. Jelaskan kerangka C++.

MODUL 2
Tipe Data, Operator C++, Struktur C++

A. Tujuan

Mahasiswa mampu mengenal tipe bilangan bulat (integer), tipe bilangan desimal (floating-point), tipe logika (boolean), tipe karakter / string, tipe data bentukan, struktur, enumerasi, operator assignment, operator unary, operator binary, operator ternary, struktur Bahasa C++, include, fungsi main(), tanda semicolon, komentar, mengenal input/output

B. Uraian Materi

Materi 1 : Tipe Data

Tipe data merupakan bagian program yang paling penting karena tipe data mempengaruhi setiap instruksi yang akan dilaksanakan oleh computer. Misalnya saja 5 dibagi 2 bisa saja menghasilkan hasil yang berbeda tergantung tipe datanya.

Jika 5 dan 2 bertipe integer maka akan menghasilkan nilai 2, namun jika keduanya bertipe float maka akan menghasilkan nilai 2,5.Tipe data berfungsi untuk merepresentasikan jenis dari sebuah nilai yang terdapat dalam program.

Dalam C++ terdapat beberapa tipe data dasar yang telah didefinisikan yaitu:

1. Tipe bilangan bulat (integer)

   Digunakan untuk data-data angka yang tidak mengandung angka dibelakang koma.

   Misalnya 3, 21, 78.

   contoh:

```
#include <iostream.h>
int main()
{
        int x; //mendeklarasikan variabel x dengn tipe data int
        x=3; //melakukan assigment terhadap variabel x
        cout <<"Nilai x adalah "<<x;
        return 0;
}
```

2. Tipe Bilangan Desimal (floating-point)

Tipe yang mempresentasikan data-data bilangan yang mengandung angka dibelakang koma, misalnya 3.16, 21.5, dll.

Contoh

Tipe data Float = 32, double = 64, dan long double = 80.

```
#include <iostream.h>
int main()
{
        double y;
        y = 27.55; //melakukan assigment terhadap variabel y
        cout <<"Nilai y adalah "<<y;
        return 0;
}
```

3. Tipe Logika (boolean)

Tipe ini merepresentasikan data-data yang hanya mengandung dua buah nilai, yaitu nilai logika (boolean) yang terdiri dari nilai benar (direpresentasikan dengan nilai 1) dan nilai salah (direpresentasikan dengan nilai 0).

4. Tipe Karakter / String

Tipe ini merepresentasikan data-data yang berupa karakter, dan dinyatakan dengan tipe char, sedangkan untuk string dinyatakan dengan pointer dari tipe char yaitu char*.

Contoh :

```
#include <iostream.h>
int main()
{
char Karakter ='A';
char*nama ="Susi Susanti";
char Jurusan[15]="INFORMATIKA";
cout <<Karakter<<endl;
cout <<nama<<endl;
cout <<Jurusan<<endl;
return 0;
}
```

5. Tipe data Bentukan

Tipe data yang dibuat sendiri sesuai kebutuhan dalam program yang akan dimuat.

a. Struktur

Struktur adlah tipe data bentukan yang menyimpan lebih dari satu variabel bertipe sama maupun berbeda.

Deklarasinya:

```
Struct nama_struktur
{
Tipe_data variabel1;
Tipe_data variabel2;
.....
};
```

b. Enumerasi

Tipe Enumerasi adalah tipe data yang nilainya terbatas pada nilai-nilai yang telah didefinisikan saja.

Tipe enumerasi digunakan untuk membentuk tipe data yang nilainya pasti.

Deklarasinya:

enum nama_tipe*{nilai1, nilai2,....};*

Contoh pendefinisian tipe enumerasi:

enum JENIS_KELAMIN{Pria, Wanita};

enum HARI{Minggu, Senin, Selasa, Rabu, Kamis, Jumat, Sabtu};

Materi 2 : Operator C++

- Operator adalah simbol yang mengolah nilai pada operand dan menghasilkan satu nilai baru.
- Operator dapat dikelompokan menjadi 4 bagian yaitu:
  1. Operator Assignment
  2. Operator Unary
  3. Operator Binary
  4. Operator Ternary

1. Operator Assignment

Operator yang berfungsi untuk memasukkan (assign) nilai ke dalam suatu variabel ataupun konstanta. Operator ini dilambangkan dengan tanda sama dengan (=)

Contoh:

```
MyChar = 'C';
MyString = "Rahasia C++";
MyInteger = 24;
MyDouble = 22.021;
```

2. Operator Unary

Operator yang hanya melibatkan sebuah operand. Yang termasuk ke dalam table operator unary antara lain:

Operator Jenis Operasi

Contoh

+ Positif +7

- Negatif -7

++ Incremen C++

Increment adalah suatu penambahan nilai yang terjadi pada sebuah variabel. Operator yang digunakan untuk melakukan increment adalah operator ++.

Ada dua jenis *increment* dalam C++ yaitu :

Pre-increment artinya melakukan penambahan nilai sebelum suatu variabel itu diproses.
Post-incremen artinya melakukan proses terlebih dahulu sebelum dilakukan penambahan nilai
Decrement merupakan kebalikan dari proses *increment*, yaitu menurunkan (mengurangi) nilai dari suatu variabel.

3. Operator Binary
Operator yang digunakan dalam operasi yang melibatkan dua buah operand.
Operator Binary dikelompokan menjadi 4 jenis yaitu:
a. Operator Aritmatika
b. Operator Logika
c. Operator Relasional
d. Operator Bitwise

a. Operator Aritmatika
Operator yang digunakan untuk melakukan operasi-operasi aritmatika seperti penjumlahan, pengurangan, dan sebagainya.

Operator	Keterangan
*	Perkalian
/	Pembagian
%	Modulus atau sisa bagi
+	Penjumlahan
-	Pengurangan

b. Operator Logika
Operator yang digunakan untuk melakukan operasi dimana nilai yang dihasilkan dari operasi tersebut hanya bernilai benar (true / 1) atau salah (false / 0).
Nilai ini biasa disebut dengan boolean.

1. Operator AND (&&)
   menghasilkan nilai 1 (benar) jika semua operand-nya bernilai benar, jika tidak maka operasi tersebut akan menghasikan nilai 0 (salah).
2. Operator OR (||)
   menghasilkan nilai 0 (salah) jika semua operand-nya bernilai salah, namun jika tidak maka operasi tersebut akan menghasilkan nilai 1 (benar).
3. Operator NOT (!)
   Nilai yang dihasilkan oleh nilai NOT adalah kebalikan dari nilai yang dikandung di dalamnya. Jika nilai awal 1 (benar) maka nilai operasi NOT menjadi 0 (salah).

c. Operator Relasional
Operator yang digunakan untuk menentukan relasi atau hubungan dari dua buah operand. Operator ini banyak digunakan untuk melakukan pengecekan sebuah ekspresi (kondisi) dalam struktur percabangan.

Operator	Jenis Operasi
>	Lebih besar
<	Lebih kecil
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan
==	Sama dengan
!=	Tidak sama dengan

d. Operator Bitwise
Operator yang digunakan untuk melakukan operasi-operasi yang berhubungan dengan pemanipulasian bit.

Operator	Keterangan
<<	Shift left
>>	shift right
&&	operasi bit AND
|	Operasi bit OR
^	Operasi bit XOR
~	Operasi bit NOT

4. Operator Ternary

Operator yang digunakan dalam operasi yang melibatkan tiga buah operand.
Bentuk umum:

Ekspresi1? Ekspresi2: Ekspresi3;

Jika ekspresi1 bernilai benar, maka program akan mengeksekusi ekspresi2. Sedangkan jika ekspresi1 salah maka yang dieksekusi adalah ekspresi3.

Materi 3 : Struktur Bahasa C++

1. include

Salah satu Pengarah Preprosesor *(preprocessor directive)* yang tersedia pada C++.
Bentuk umumnya :

include <nama_file>

tidak diakhiri dengan tanda semicolon (;), karena bentuk tersebut bukanlah suatu bentuk pernyataan, tetapi merupakan prepocessor directive.

2.Fungsi main ()

Program C++ terdiri dari satu atau lebih fungsi, dan di antara salah satunya harus ada fungsi main dan hanya boleh ada satu main pada tiap program C++.

3. Komentar

Komentar tidak pernah dicompile oleh compiler. Dalam C++ terdapat 2 jenis komentar, yaitu:

Jenis 1 : /* Komentar anda diletakkan di dalam ini
Bisa mengapit lebih dari satu baris */
Jenis 2 : // Komentar anda diletakkan disini (hanya bisa sebaris)

4.Tanda Semicolon

Tanda semicolon " ; " digunakan untuk mengakhiri sebuah pernyataan. Setiap pernyataan harus diakhiri dengan sebuah tanda semicolon.

5. Mengenal Input/Output

Pernyataan cout (dibaca C out) merupakan sebuah objek di dalam C++, yang digunakan untuk mengarahkan data ke dalam standar output (cetak pada layar). Sedangkan untuk menginputkan data, dapat digunakan cin (dibaca C in).

D. Rangkuman

- Tipe data : bilangan bulat (integer), bilangan desimal (floating-point), logika (boolean), karakter / string
- Operator C++ : operator assignment, operator unary, operator binary, operator ternary

D. Tugas

1. Membuat resume dari materi yang telah diterangkan.
2. Tanya jawab dan diberikan latihan soal terhadap individu.

E. Test

1. Buatkan contoh bilangan bulat (integer) dan bilangan desimal (floating-point).
2. Buatkan contoh penggunaan operator assignment dan operator binary.
3. Jelaskan tipe data bentukan.

MODUL 3

STRUKTUR, ARRAY, POINTER

A. Tujuan

Mahasiswa mampu memahami struktur, mendeklarasikan struktur, defenisi struktur, bentuk umum sttruktur, penjelasan array, defenisi array, karakteristik array, penjelasan array satu dimensi, bentuk umum array satu dimensi, inisialisasi array satu dimensi, penjelasan array dua dimensi, bentuk umum array dua dimensi, inisialisasi array dua dimensi, penjelasan array tiga dimensi, bentuk umum array tiga dimensi, inisialisasi array tiga dimensi, deklarasi pointer.

B. Uraian Materi

Materi 1 : Struktur

Struktur adalah koleksi dari variabel yang dinyatakan dengan sebuah nama, dengan sifat setiap variabel dapat memiliki tipe yang berlainan. Struktur biasa dipakai untuk mengelompokkan beberapa informasi yang berkaitan menjadi sebuah satu kesatuan.

Mendeklarasikan Struktur

Contoh pendefinisian tipe struktur adalah sebagai berikut:

```
struct data_tanggal
{
int tanggal;
int bulan;
int tahun;
};
```

yang mendefinisikan tipe struktur bernama data_tanggal, yang terdiri dari tiga buah elemen (field) berupa : tanggal, bulan dan tahun.

Pendefnisian dan pendeklarasian struktur dapat juga ditulis sebagai berikut:

```
struct data_tanggal
{
int tanggal;
int bulan;
int tahun;
} tgl_lahir;
```

Bentuk umum dalam mendefinisikan dan mendeklarasikan struktur adalah sebagai berikut :

```
struct nama_tipe_struktur
{
tipe field1;
tipe field2;
.
.
tipe fieldn;
}variabel_struktur1, ... , variabel_strukturM;
```

Masing-masing tipe dari elemen struktur dapat berlainan. Adapun variabel_struktur1 sampai dengan variabel_strukturM menyatakan bahwa variabel struktur yang dideklarasikan bisa lebih dari satu. Jika ada lebih dari satu variabel, antara variable struktur dipisahkan dengan tanda koma.

Mengakses Elemen Struktur

Elemen dari struktur dapat diakses dengan menggunakan bentuk variabel_struktur.nama_field
Antara variabel_struktur dan nama_field dipisahkan dengan operator titik (disebut operator anggota struktur). Contoh berikut merupakan instruksi untuk mengisikan data pada field tanggal
tgl_lahir.tanggal = 30;

Materi 2 : Array

- Pengertian array
  - Variabel larik disebut array adalah tipe terstruktur yang terdiri dari sejumlah komponen-komponen yang sama.
  - Sekelompok data yang sejenis yang disimpan didalam memori secara berurutan
  - Jumlah komponen : tetap

- Banyak komponen dalam sebuah larik ditentukan sebuah indeks.
- Indeks membedakan variabel yang satu dengan yang lain.
- Setiap elemen mempunyai nilai indek sesuai dengan urutannya.
- Indek dari elemen array ini, baik dalam bahasa c++ maupun java dimulai dari 0, bukan 1 seperti dalam bahasa pascal.
- Array dideklarasikan dengan tanda [ ] (bracket), baik dalam bahasa C++ dan Java.
- Array (larik) ialah penampung sejumlah data sejenis (homogen) yang menggunakan satu identifier (pengenal).
- Masing-masing elemen larik diakses menggunakan indeks (subscript) dari nol sampai n-1 (n menyatakan jumlah elemen larik).
- Pengolahan data larik harus per elemen. Elemen larik dapat diakses langsung (acak), maksudnya memanipulasi elemen keempat tidak harus melalui elemen kesatu, kedua dan ketiga.

- Defenisi array
  sebagai suatu himpunan elemen, terurut dan homogen.
  pemesanan alokasi memory sementara pada komputer.
  Terurut : Dapat diartikan bahwa elemen tersebut dapat diidentifikasi sebagai elemen pertama, elemen kedua dan seterusnya sampai elemen ke-n.
  Homogen : setiap elemen darisebuah Array tertentu haruslah mempunyai type data yang sama.
- Karakteristik Array :
  Mempunyai batasan dari pemesanan alokasi, memory (bersifat statis).
  Mempunyai type data sama, (bersifat homogen).
  Dapat diakses secara acak
- Bentuk umum dari tipe data array adalah :
  tipe_data nama_array[jumlah_elemen]
  Contoh deklarasi array :
  Int arr [5] ; atau double d [10] ;
- Array terdiri dari :
  Array Satu Dimensi
  Array Dua Dimensi
  Array Multidimensi

Inisialisasi array

elemen-elemen array diletakkan diantara tanda kurung.

Contoh :

```
Int arr [ 5 ] = { 1, 3, -3, 5, 2 } ;
```

Jika jumlah elemen yang diinisialisasikan kurang dari jumlah elemen yang tersedia, maka sisa elemen tersebut akan diberikan nilai 0 (nol) secara otomatis oleh compiler

Contoh Program

```
# include <iostream.h>
Main ( )
{
Int arr [ 5 ] = { 1, 3 } ;
For ( int i = 0 ; i < 5 ; i++ )
Cout << " arr [ i ] << ' ' ;
Return 0 ;
}
```

Output : 1 3 0 0 0

Array Satu Dimensi

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

Bentuk umum :

```
Tipe Data Nama_Variabel [Ukuran]
```

Keterangan :
Type Data : untuk menyatakan type data yang digunakan
Ukuran: menyatakan maksimum array maksimum elemen array

Sebelum variabel array digunakan maka variabel array harus dideklarasikan terlebih dahulu. Pendeklarasian variabel array satu dimensi sebenarnya hampir sama dengan pendeklarasian variabel yang lain , hanya saja pendeklarasian variabel array diikuti dengan maksimum banyaknya elemen yang dapat disimpan dalam variabel array yang dituliskan dalam pasangan tanda siku penutup. Di dalam bahasa C++, harga awal indeks dimulai dari [nol]. Maka jika dituliskan banyaknya maksimum elemen adalah N, berarti indexs yang akan digunakan adalah 0,1,2 . . . . N-1.

Bentuk umum:

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

n = jumlah elemen

Mengakses Array Dimensi Satu

Inisialisasi Array Dimensi Satu

inisialisasi adalah memberikan nilai awal terhadap suatu variabel.

bentuk pendefenisian Array Dimensi Satu ::

Bentuk umum larik satu dimensi dideklarasikan dengan tipe_data menyatakan jenis elemen larik (int, float, char, unsigned, dan lain-lain), tidak berjenis void.

nama_array adalah nama array, harus memenuhi ketentuan pengenal.

ukuran menyatakan jumlah maksimal elemen array, normalnya lebih besar dari satu.

Menghitung Jumlah Elemen Array

fungsi sizeof() mengembalikan jumlah byte yang sesuai dengan argumennya, maka operator tersebut dapat digunakan untuk menemukan jumlah elemen array

Misalnya

```
int array[ ] = {26,7,82,166};
cout<<sizeof(array)/sizeof(int);
```

akan mengembalikan nilai 4, yaitu sama dengan jumlah elemen yang dimiliki array.

Array Sebagai Argumen Fungsi

Array dapat dikirim dan dikembalikan oleh fungsi pada saat array dikirim ke dalam fungsi, nilai aktualnya dapat dimanipulasi.

Contoh :

```
#include <iostream.h>
void ubah(int x[]);
void main()
{
int ujian[] = {90,95,78,85};
ubah(ujian);
cout<<" Elemen kedua dari array ujian adalah "<<ujian[1]<<endl;
}
void ubah(int x[])
{
x[1] = 100;
}
```

Keluarannya : Elemen kedua dari array ujian adalah 100

Array Dua Dimensi

Mengakses Array Dimensi Dua

Contoh :

data_jual [2] [2];

data_jual [1] [2];

Array Dua Dimensi : Matriks

Matriks adalah sekumpulan informasi yang setiap individu elemennya diacu dengan menggunakan dua buah indeks (yang biasanya dikonotasikan dengan baris dan kolom).

Suatu array dapat diakses dengan menggunakan subscript atau index.

Bentuk umum :

```
Tipe Data Nama_Array[index-1]
```

Keterangan :
Type Data : untuk menyatakan type data yang digunakan
Index-1: menyatakan jumlah baris
Index-2 : menyatakan jumlah isi dan baris

Konsep umum larik berlaku juga pada Matriks, yaitu:

1. Kumpulan elemen bertipe sama;
2. Setiap elemen data dapat diakses secara acak, jika indeksnya (baris dan kolom) sudah diketahui;
3. Merupakan struktur data statis, artinya jumlah elemennya sudah ditentukan terlebih dahulu di dalam kamus dan tidak bisa diubah selama pelaksanaan program.

Deklarasi Matriks :

tipe_data nama_matriks[baris] [kolom];

Inisialisasi Matriks

Menjumlahkan Dua Buah Matriks

Menjumlahkan dua buah matriks A dan B menghasilkan matriks C atau A + B = C

hanya dapat dilakukan bila ukuran matriks A dan ukuran matriks B sama, dan kedua matriks sudah terdefinisi harganya (elemennya sudah terisi).

Matriks C juga berukuran

sama dengan matriks A dan B. Penjumlahan matriks A dan B didefinisikan sebagai

Inisialisasi Array Dua Dimensi

Contoh Pendeklarasian Array 2 Dimensi

Array Tiga Dimensi

Array dimensi dua tersusun dalam bentuk baris, kolom dan isi dari baris, indeks pertama menunjukkan baris, indeks kedua menunjukkan kolom dan indeks ketiga menunjukkan isi dan baris

Bentuk umum pendeklarasian array :

```
Tipe-Data Nama_Variabel[index-1] [Index-2] [Index-3]
```

Keterangan :
Type Data : untuk menyatakan type data yang digunakan
Index-1: menyatakan jumlah baris
Index-2 : menyatakan jumlah isi dan baris
Index-3 : menyatakan kolom

Materi 3 : Pointer

Misalnya kita ingin membuat beberapa penunjuk ke blok penyimpan yang berisi integer.
Deklarasi pada C adalah:

```
int *IntegerPointer;
```

Tanda asterik (*) yang berada sebelum nama variable IntegerPointer menandakan ‘pointer pada suatu int’. Jadi deklarasi diatas berarti ‘definisikan sebuah tipe yang terdiri dari pointer bertipe integer yang bernama IntegerPointer’.

Apabila didepannya ditambahkan typedef sebagai berikut :

```
Typedef int *IntegerPointer;
```

Berarti IntegerPointer merupakan suatu tipe pointer berbentuk integer.
Apabila akan mendeklarasikan dua variable A dan B sebagai penunjuk ke bilangan integer :

```
IntegerPointer A, B;
```

Berarti kompiler C akan berisi nilai dari variable A dan B yang 'menunjuk ke integer'.
Untuk membuat beberapa penunjuk ke beberapa penyimpan integer yang kosong dan untuk membuat A dan B menunjuk tempat tersebut, digunakan prosedur dinamis untuk alokasi penyimpan yang disebut malloc.

C. Rangkuman

1. Struktur data adalah sebuah skema organisasi yang diterapkan pada data sehingga data dapat diinterprestasikan dan sehingga operasi-operasi spesifik dapat dilaksanakan pada data tersebut.
2. Apabila membuat program dengan data yang sudah diketahui batasnya, maka bisa menggunakan array (tipe data statis), namun apabila data belum diketahui batasnya, bisa menggunakan pointer (tipe data dinamis).
3. Untuk sekumpulan data dengan tipe data yang berlainan, namun merupakan satu-kesatuan, dapat menggunakan struktur untuk merepresentasikannya.

D. Tugas

1. Membuat resume dari materi yang telah diterangkan.
2. Tanya jawab dan diberikan latihan soal terhadap individu.

E. Test

1. Jelaskan bentuk umum array 1, 2, 3 dimensi.
2. Buatkan deklarasi struktur.
3. Jelaskan defenisi pointer.

MODUL 4
PENGURUTAN DATA (*SORTING*), PENCARIAN *(SEARCHING)*

A. Tujuan

Mahasiswa mampu memahami defensi pengurutan data (*sorting*), urutan proses pengurutan, kategori pemilihan algoritma, deklarasi larik, Metode Penyisipan Langsung (*Straight Insertion Sort*), Metode Penyisipan Langsung (*Straight Insertion Sort*), Metode Penyisipan Biner (*Binary Insertion Sort*), Metode Seleksi (*Selection Sort*), Metode Gelembung (*Bubble sort*), Metode Shell (Shell Sort), Metode Quick (Quick Sort), Metode Quick Sort Non Rekursif, Metode Quick Sort Rekursif, Metode Penggabungan (Merge Sort).

B. Uraian Materi

Materi 1 : Defenisi Pengurutan data (*sorting*)

Pengurutan data (*sorting*) didefinisikan sebagai suatu proses untuk menyusun kembali humpunan obyek menggunakan aturan tertentu.

Menurut Microsoft Book-shelf, definisi algoritma pengurutan adalah algoritma untuk meletakkan kumpulan elemen data ke dalam urutan tertentu berdasarkan satu atau beberapa kunci dalam tiap-tiap elemen.

Ada dua macam urutan yang biasa digunakan dalam proses pengurutan yaitu

• urut naik (*ascending*) yaitu dari data yang mempunyai nilai paling kecil sampai paling besar.

• urut turun (*descending*) yaitu data yang mempunyai nilai paling besar sampai paling kecil.

Contoh : data bilangan 5, 2, 6 dan 4 dapat diurutkan naik menjadi 2, 4, 5, 6 atau diurutkan turun menjadi 6, 5, 4, 2. Pada data yang bertipe char, nilai data dikatakan lebih kecil atau lebih besar dari yang lain didasarkan pada urutan relatif (*collating sequence*) seperti dinyatakan dalam tabel ASCII.

Keuntungan dari data yang sudah dalam keadaan terurutkan antara lain :

- Data mudah dicari (misalnya dalam buku telepon atau kamus bahasa), mudah untuk dibetulkan, dihapus, disisipi atau digabungkan. Dalam keadaan terurutkan, mudah melakukan pengecekan apakah ada data yang hilang.
- Melakukan kompilasi program komputer jika tabel-tabel simbol harus dibentuk.
- Mempercepat proses pencarian data yang harus dilakukan berulang kali.

Beberapa faktor yang berpengaruh pada efektifitas suatu algoritma pengurutan antara lain:

• banyak data yang diurutkan.

• kapasitas pengingat apakah mampu menyimpan semua data yang dimiliki

• tempat penyimpanan data, misalnya piringan, pita atau kartu, atau media penyimpan yang lain.

Metode pengurutan yang digunakan dapat diklasifikasikan menjadi dua kategori yaitu :

- Pengurutan internal, yaitu pengurutan dengan menggunakan larik (*array*). Larik tersimpan dalam memori utama komputer
- Pengurutan eksternal, yaitu pengurutan dengan menggunakan berkas (*sequential access file*). Berkas tersimpan dalam pengingat luar, misalnya cakram atau pita magnetis.

Deklarasi Larik

Pada pengurutan larik, ada beberapa aspek yang perlu dipertimbangkan, antara lain aspek menyangkut kapasitas pengingat yang ada dan aspek waktu, yaitu waktu yang diperlukan untuk melakukan permutasi sehingga semua elemen akhirnya menjadi terurutkan.

Deklarasi larik yang digunakan adalah larik dimensi satu (*vektor*) dengan elemennya bertipe integer.

Materi 2 : Metode Penyisipan Langsung (*Straight Insertion Sort*)

Proses pengurutan dengan metode penyisipan langsung dapat dijelaskan sebagai berikut :

Data dicek satu per satu mulai dari yang kedua sampai dengan yang terakhir. Apabila ditemukan data yang lebih kecil daripada data sebelumnya, maka data tersebut disisipkan pada posisi yang sesuai. Akan lebih mudah apabila membayangkan pengurutan kartu. Akan lebih mudah apabila membayangkan

pengurutan kartu. Pertama-tama 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 penyisipan langsung dapat dituliskan sebagai berikut :

```
1  i ← 1
2  selama (i < N) kerjakan baris 3 sampai dengan 9
3  x ← Data[i]
4   j ← i – 1
5  selama (x < Data[j]) kerjakan baris 6 dan 7
6  Data[j + 1] ← Data[j]
7  j ← j – 1
8  Data[j+1] ← x
9  i ← i + 1
```

Proses pengurutan :

i=1, x sama dengan Data[1] = 35 dan j=0. Karena Data[0] = 12 dan 35 > 12 maka proses dilanjutkan untuk i=2 ; i=2, x = Data[2] = 9 dan j=1. Karena Data[1] = 35 dan 9 < 35, maka dilakukan pergeseran sampai ditemukan data yang lebih kecil dari 9. Hasil pergeseran ini, Data[1] = 12 dan Data[2] = 35 sedangkan Data[0] = x = 9 ; i=3, x = Data[3] = 11 dan j=3. Karena Data[2] = 35 dan 11 < 35, maka dilakukan pergeseran sampai ditemukan data yang lebih kecil dari 11. Hasil pergeseran ini, Data[2] = 12 dan Data[3] = 35 sedangkan Data[1] = x = 11. dan seterusnya.

Materi 3 : Metode Penyisipan Biner (*Binary Insertion Sort*)

Metode ini merupakan pengembangan dari metode penyisipan langsung. Dengan cara penyisipan langsung, perbandingan selalu dimulai dari elemen pertama (data ke-0), sehingga untuk menyisipkan elemen ke i kita harus melakukan perbandingan sebanyak i-1 kali. Metode ini didasarkan pada kenyataan bahwa pada saat menggeser data ke-i, data ke 0 s/d i-1 sebenarnya sudah dalam keadaan terurut.

Materi 4 : Metode Seleksi (Selection Sort)

Metode seleksi melakukan pengurutan dengan cara mencari data yang terkecil kemudian menukarkannya dengan data yang digunakan sebagai acuan atau sering dinamakan pivot. Proses pengurutan dengan metode seleksi dapat dijelaskan sebagai berikut : langkah pertama dicari data terkecil dari data pertama sampai data terakhir. Kemudian data terkecil ditukar dengan data pertama. Dengan demikian, data pertama sekarang mempunyai nilai paling kecil dibanding data yang lain. Langkah kedua, data terkecil.

Materi 5 : Metode Gelembung (*Bubble sort*)

Metode gelembung (*bubble sort*) sering juga disebut dengan metode penukaran (*exchange sort*) adalah metode yang mengurutkan data dengan cara membandingkan masing-masing elemen, kemudian melakukan penukaran bila perlu. Metode ini mudah dipahami dan diprogram, tetapi bila dibandingkan dengan metode lain yang kita pelajari, metode ini merupakan metode yang paling tidak efisien. Proses pengurutan metode gelembung ini menggunakan dua kalang. Kalang pertama melakukan pengulangan dari elemen ke 2 sampai dengan elemen ke N-1 (misalnya variable i), sedangkan kalang kedua melakukan pengulangan menurun dari elemen ke N sampai elemen ke i (misalnya variable j). Pada setiap pengulangan, elemen ke j-1 dibandingkan dengan elemen ke j. Apabila data ke j-1 lebih besar daripada data ke j, dilakukan penukaran.

Materi 6 : Metode Shell (Shell Sort)

Metode ini disebut juga dengan metode pertambahan menurun (diminishing increment). Metode ini dikembangkan oleh Donald L. Shell pada tahun 1959, sehingga sering disebut dengan Metode Shell Sort. Metode ini mengurutkan data dengan cara membandingkan suatu data dengan data lain yang memiliki jarak tertentu, kemudian dilakukan penukaran bila diperlukan.

Proses pengurutan dengan metode Shell dapat dijelaskan sebagai berikut :

Pertama-tama adalah menentukan jarak mula-mula dari data yang akan dibandingkan, yaitu N / 2. Data pertama dibandingkan dengan data dengan jarak N / 2. Apabila data pertama lebih besar dari data ke N / 2 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan dengan jarak yang sama yaitu N / 2. Demikian seterusnya sampai seluruh data dibandingkan sehingga semua data ke-j selalu lebih kecil daripada data ke-(j + N / 2). Pada proses berikutnya, digunakan jarak (N / 2) / 2 atau N / 4. Data pertama dibandingkan dengan data dengan jarak N / 4. Apabila data pertama lebih besar dari data ke N / 4 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan dengan jarak yang sama yaitu N / 4. Demikian seterusnya sampai seluruh data
dibandingkan sehingga semua data ke-j lebih kecil daripada data ke-(j + N / 4).

Materi 7 : Metode Quick (Quick Sort)

Metode Quick sering disebut juga metode partisi (partition exchange sort). Metode ini diperkenalkan pertama kali oleh C.A.R. Hoare pada tahun 1962. Untuk mempertinggi efektifitas dari metode ini, digunakan teknik menukarkan dua elemen dengan jarak yang cukup besar.

Proses penukaran dengan metode quick dapat dijelaskan sebagai berikut.: mulamula dipilih data tertentu yang disebut pivot, misalnya x. Pivot dipilih untuk mengatur data di sebelah kiri agar lebih kecil daripada pivot dan data di sebelah kanan agar lebih besar daripada pivot. Pivot ini diletakkan pada posisi ke j sedemikian sehingga data antara 1 sampai dengan j-1 lebih kecil daripada x. Sedangkan data pada posisi ke j+1 sampai N lebih besar daripada x. Caranya dengan menukarkan data diantara posisi 1 sampai dengan j-1 yang lebih besar daripada x dengan data diantara posisi j+1 sampai dengan N yang lebih kecil daripada x.

Metode Quick Sort Non Rekursif

Implementasi secara non rekursif memerlukan dua buah tumpukan (stack) yang digunakan yang digunakan untuk menyimpan batas-batas subbagian. Pada prosedur ini menggunakan tumpukan yang bertipe record (struktur) yang terdiri dari elemen kiri (untuk mencatat batas kiri) dan kanan (untukmencatat batas kanan. Tumpukan dalama hal ini dideklarasikan sebagai array.

Metode Quick Sort Rekursif

Algoritma quick Rekursif dapat dituliskan sebagai berikut :

```
1   x ← Data[(L + R) / 2]
2   i ← L
3   j ← R
4   Selama ( i <= j) kerjakan baris 5 sampai dengan 12
5   Selama (Data[i] < x) kerjakan i ← i + 1
6   Selama (Data[j] > x) kerjakan j ← j – 1
7   Jika (i <= j) maka kerjakan baris 8 sampai dengan 10; jika tidak kerjakan baris 11
8   Tukar Data[i] dengan Data[j]
9   i ← i + 1
10  j ← j –1
11  Jika (L < j) kerjakan lagi baris 1 dengan R = j
12  Jika (i < R) kerjakan lagi baris 1 dengan L = i
```

Materi 8 : Metode Penggabungan (Merge Sort)

Metode penggabungan biasanya digunakan pada pengurutan berkas. Prinsip dari metode penggabungan sebagai berikut : mula-mula diberikan dua kumpulan data yang sudah dalam keadaan urut. Kedua kumpulan data tersebut harus dijadikan satu table sehingga dalam keadaan urut.

Materi 9 : Searching (Pencarian)

Pencarian data sering juga disebut *table look-up* atau *storage and retrieval information* adalah suatu proses untuk mengumpulkan sejumlah informasi di dalam pengingat komputer dan kemudian mencari kembali informasi yang diperlukan secepat mungkin.

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

Metode pencarian data dapat dilakukan dengan dua cara yaitu **pencarian internal** (*internal searching*) dan **pencarian eksternal** (*external searching*). Pada pencarian internal, semua rekaman yang diketahui berada dalam pengingat komputer sedangakan pada pencarian eksternal, tidak semua rekaman yang diketahui berada dalam pengingat komputer, tetapi ada sejumlah rekaman yang tersimpan dalam penyimpan luar misalnya pita atau cakram magnetis.

Metode pencarian data : **pencarian statis** (*static searching*) dan **pencarian dinamis** (*dynamic searching*). Pada pencarian statis, banyaknya rekaman yang diketahui dianggap tetap, pada pencarian dinamis, banyaknya rekaman yang diketahui bisa berubah-ubah yang disebabkan oleh penambahan atau penghapusan suatu rekaman.

Ada dua macam teknik pencarian yaitu pencarian sekuensial dan pencarian biner. Perbedaan dari dua teknik ini terletak pada keadaan data. **Pencarian sekuensial** digunakan apabila data dalam keadaan acak atau tidak terurut. Sebaliknya, **pencarian biner** digunakan pada data yang sudah dalam keadaan urut.

C. Rangkuman

1. Metode Bubble Sort sekalipun relatif lebih mudah dilakukan tetapi tidak efisien dibandingkan dengan metode lain dalam melakukan pengurutan.
2. Metode Quick Sort dan metode Merge Sort dapat dilakukan dengan menggunakan rekursif atau tanpa rekursif.

D. Tugas

1. Membuat resume dari materi yang telah diterangkan.
2. Tanya jawab dan diberikan latihan soal terhadap individu.

E. Test

1. Tuliskan kategori pemilihan algoritma.
2. Jelaskan algoritma quick Rekursif.
3. Jelaskan perbedaan Metode Quick Sort Non Rekursif dengan Metode Quick Sort Rekursif.
4. Tuliskan contoh Merge Sort.

MODUL 5

PERULANGAN (LOOPING), PERCABANGAN (DECISION)

A. Tujuan

Mahasiswa mampu memahami pernyataan while, bentuk umum while, pernyataan Do… While, bentuk umum Do… While… , pernyataan For, bentuk umum For, pernyataan For Bersarang (Nested For), bentuk umum For Bersarang (Nested For), decision (percabangan), Struktur satu kondisi (perintah IF), Struktur satu kondisi (perintah IF), Struktur dua kondisi (perintah if – else), Struktur tiga kondisi (perintah multiple if – else), perintah switch

B. Uraian Materi

Materi 1 : Pernyataan While

Pada pernyataan while, pengecekan terhadap loop dilakukan dibagian awal (sebelum tubuh loop).
Bentuk umum :

```
while (kondisi)
pernyataan;
```

Perulangan ini banyak digunakan bila jumlah perulangannya belum diketahui. Proses perulangan akan terus berlanjut selama kondisinya bernilai benar (true) dan akan berhenti bila kondisinya bernilai salah.

Materi 2 : Pernyatan Do .. While…

Pada pernyataan do-while, tubuh loop berupa pernyataan. Pada pernyataan do, mula-mula pernyataan dijalankan. Selanjutnya, kondisi diuji. Seandainya kondisi bernilai benar, maka pernyataan dijalankan lagi, kemudian kondisi diperiksa kembali, dan seterusnya. Kalau kondisi bernilai salah, maka pernyataan tidak dijalankan lagi. Pada dasarnya struktur perulangan do....while sama saja dengan struktur while, hanya saja pada proses perulangan dengan while, seleksi berada di while yang letaknya di atas sementara pada perulangan do....while, seleksi while berada di bawah batas perulangan. Jadi dengan menggunakan struktur do…while sekurang-kurangnya akan terjadi satu kali perulangan.

Bentuk umum:

```
do
        pernyataan;
while (kondisi)
```

Materi 3. Pernyataan For

Struktur perulangan for biasa digunakan untuk mengulang suatu proses yang telah diketahui jumlah perulangannya. Dari segi penulisannya, struktur perulangan for tampaknya lebih efisien karena susunannya lebih simpel dan sederhana.
Bentuk umum perulangan for:

```
for(inisialisasi; syarat; penambahan)
pernyataan;
```

Inisialisasi : pernyataan untuk menyatakan keadaan awal dari variabel kontrol. *syarat* : ekspresi relasi yang menyatakan kondisi untuk keluar dari perulangan. *penambahan* : pengatur perubahan nilai variabel kontrol.

Materi 4 : For Bersarang (Nested For)

For bersarang dapat melibatkan lebih dari satu variabel namun yang jelas satu diantaranya akan digunakan sebagai indeks perulangan.
Bentuk umum:

```
for( variabel1=nilai_awal; kondisi1; variabel1++){
        for( variabel2=nilai_awal; kondisi2; variabel2++){
                for( variabel3=nilai_awal; kondisi3; variabel3++){
                        Statemen_statemen yang akan diulang;
                                ....
                }
```

```
        }
}
```

Materi 4 : *Decision* (Perulangan)

Decision digunakan untuk memilih salah satu alternatif jawaban yang tepat dari pilihan yang ada.
Suatu pemilihan statemen yang akan dieksekusi dimana pemilihan tersebut didasarkan atas kondisi tertentu. Statemen yang terdapat dalam sebuah blok percabangan akan dieksekusi jika kondisi yang didefinisikan terpenuhi (bernilai benar) Artinya jika kondisi tidak terpenuhi (bernilai salah) maka statemen tersebut tidak ikut dieksekusi atau akan diabaikan oleh **compiler.**

1. Struktur satu kondisi (perintah IF)

- digunakan untuk menyeleksi suatu kondisi tunggal.
- Bila proses yang diseleksi terpenuhi atau bernilai benar, maka pernyataan yang ada di dalam blok if akan diproses dan dikerjakan.

Bentuk umum struktur kondisi if adalah:

```
if(kondisi)
```

2. Struktur dua kondisi (perintah if – else)

- Perintah if.....else minimal terdapat dua pernyataan.
- Jika kondisi yang diperiksa bernilai benar atau terpenuhi maka pernyataan pertama yang dilaksanakan dan jika kondisi yang diperiksa bernilai salah maka pernyataan yang kedua yang dilaksanakan.

Bentuk Umum

```
if (kondisi)
        {
statemen_jika_kondisi_terpenuhi;
        }
else
        {
statemen_jika_kondisi_tidak_terpenuhi;
        }
```

3. Struktur tiga kondisi (perintah multiple if – else)

Percabangan jenis ini merupakan perluasan dari struktur yang memiliki dua kondisi diatas yaitu dengan menyisipkan satu atau lebih kondisi ke dalamnya.

Bentuk umum:

```
if (kondisi1)
{
statemen_jika_kondisi1_terpenuhi;
}
        else if (kondisi2)
        {
statemen_jika_kondisi2_terpenuhi;
        }
else if (kondisi3)
{
statemen_jika_kondisi3_terpenuhi;
}
....
else
{
statemen_jika_semua_kondisi_tidak_terpenuhi;
}
```

4. Perintah switch

Perintah ini memiliki bentuk switch – case yang digunakan untuk pilihan berjumlah banyak. Perintah switch tidak dianjurkan pada pilihan yang melibatkan jangkauan nilai (range) tetapi dianjurkan pada pilihan berupa konstanta dan banyak misalnya untuk memilih menu.

Bentuk umum:

```
switch (pernyataan)
{
case nilai_1:
        blok_pernyataan1;
        break;
case nilai_2:
        blok_pernyataan2;
```

```
            break;
    default:
    blok_pernyataan_n;
    }
```

Cara kerjanya:

1. switch akan mengevaluasi pilihan dan apabila isinya sama dengan nilai_1, maka blok pernyataan 1 akan dijalankan sampai menemukan perintah break untuk kemudian keluar dari blok switch.
2. Bila pilihan tidak sama isinya dengan nilai_1, maka akan dicocokkan lagi dengan nilai_2. dan apabila isinya sama dengan nilai_2, maka blok pernyataan 2 akan dijalankan sampai menemukan perintah break untuk kemudian keluar dari blok switch.
3. Apabila isi pilihan tidak sesuai dengan nilai_1, nilai_2 dan seterusnya maka secara otomatis yang dijalankan adalah blok pernyataan default.

C. Rangkuman

- Satu kondisi menggunakan perintah if, dua kondisi menggunakan perintah if – else, tiga kondisi menggunakan perintah multiple if – else).
- Perintan *Decision* digunakan untuk memilih salah satu alternatif jawaban yang tepat dari pilihan yang ada.
- Perintah *Looping* menggunakan pernyataan **While** bila perulangan satu benar, **Do.. While** berupa pernyataan, **For** untuk suatu proses yang telah diketahui jumlah perulangannya, **Nested For** melibatkan lebih dari satu variabel namun yang jelas satu diantaranya akan digunakan sebagai indeks perulangan.

D. Tugas

1. Buatlah program untuk menentukan huruf, dengan ketentuan sebagai berikut:
   Jika karakter >='A' dan karakter <='Z' maka Karakter yang Anda masukkan adalah huruf besar.
   Jika karakter >='a' dan karakter <='z' maka Karakter yang Anda masukkan adalah huruf kecil.
   Jika karakter >='0' dan karakter <='9' maka Karakter yang Anda masukkan adalah Angka.
   Jika bukan semuanya berarti Karakter yang Anda masukkan adalah bukan alphanumeric.
2. Buatlah program untuk menghitung luas dengan menggunakan menu, dengan ketentuan sebagai berikut:
   a. Menghitung Luas Bujur Sangkar.
   b. Menghitung Luas Persegi Panjang.
   c. Menghitung Luas Segi Tiga
   d. Menghitung Luas Lingkaran.

E. Test

1. Tuliskan contoh perintah perulangan multiple if - case.
2. Tuliskan contoh perintah For bersarang.
3. Tuliskan contoh perintah switch.

MODUL 6

FUNCTION

A. Tujuan

Mahasiswa mampu memahami defenisi Function, sifa-sifat dari Fungsi, bentuk umum dari Fungsi,Fungsi Tanpa Nilai Balik, Fungsi dengan Nilai Balik, fungsi Rekursif.

Materi 1 : Function

Fungsi adalah satu blok kode yang dapat melakukan tugas tertentu atau satu blok instruksi yang dieksekusi ketika dipanggil dari bagian lain dalam suatu program. Sebuah fungsi berisi sejumlah pernyataan yang dikemas dalam sebuah nama. Nama tersebut selanjutnya dapat dipanggil berkali-kali di beberapa tempat dalam program. Keuntungan pembuatan fungsi secara umum adalah :

- Program besar dapat dipisah menjadi program-program kecil.
- Dapat dikerjakan oleh beberapa orang sehingga koordinasi mudah.
- Kemudahan dalam mencari kesalahan-kesalahan karena alur logika jelas dan kesalahan dapat dilokalisasi dalam suatu modul tertentu saja.
- Modifikasi program dapat dilakukan pada suatu modul tertentu saja tanpa mengganggu program keseluruhan. Mempermudah dokumentasi.

Sifat-sifat dari fungsi yang baik adalah :

- Nilai fan-in tinggi, artinya semakin sering suatu fungsi dipanggil oleh pengguna semakin tinggi nilai fan-in.
- Fan-out rendah, artinya semakin spesifik fungsi suatu modul akan semakin rendah nilai fan-out.
- Self-contained tinggi, artinya kemampuan untuk memenuhi kebutuhannya sendiri.

Bentuk umum sebuah fungsi adalah:

```
Tipe_fungsi nama_fungsi (parameter fungsi)
Deklarasi parameter
{
Tubuh fungsi
}
```

Materi 2 : Tipe fungsi digunakan untuk menentukan tipe keluaran fungsi

I. Fungsi Tanpa Nilai Balik

Fungsi tanpa nilai balik (return Value) digunakan untuk melakukan proses-proses yang tidak menghasilkan nilai, seperti melakukan pengulangan, proses pengesetan nilai ataupun yang lainnya.
Fungsi semacam ini tipe kembaliannya akan diisi dengan nilai void.
Bentuk umumnya:

```
Void_nama_fungsi(parameter1, parameter2,..)
{
Statemen_yang_akan_dieksekusi;
. . .
}
```

2. Fungsi Dengan Nilai Balik

Fungsi dengan nilai balik yaitu fungsi yang digunakan untuk melakukan proses-proses yang berhubungan dengan nilai. Adapun cara pendefinisiannya adalah dengan menuliskan tipe data dari nilai yang akan dikembalikan didepan nama fungsi.
Bentuk umum:

```
tipe_data nama_fungsi(parameter1, parameter2,..)
{
Statemen_yang_akan_dieksekusi;
. . .
return nilai_balik;
}
```

3. Fungsi Dengan Parameter

Parameter adalah suatu variabel yang berfungsi untuk menampung nilai yang akan dikirimkan ke dalam fungsi. Parameter itu sendiri terbagi dua macam yaitu:

- Parameter formal adalah variabel yang ada pada daftar parameter dalam definisi fungsi.

- Parameter aktual adalah parameter (tidak selamanya menyatakan variabel) yang digunakan dalam pemanggilan fungsi.

Materi 3 : Rekursif

Rekursif berarti suatu proses yang memanggil dirinya sendiri. Dalam rekursif sebenarnya terkandung pengertian prosedur atau fungsi. Perbedaannya adalah bahwa rekursif bisa memanggil ke dirinya sendiri, tetapi prosedur atau fungsi harus dipanggil lewat pemanggil prosedur atau fungsi.

Rekursif merupakan teknik pemrograman yang penting, dan beberapa bahasa pemrograman modern mendukung keberadaan proses rekursif ini. Pemanggilan prosedur atau fungsi ke dirinya sendiri bisa berarti proses yang berulang yang tidak bisa diketahui kapan akan berakhir. Dalam pemakaian sehari-hari, rekursi merupakan teknik pemrograman yang berdaya guna untuk digunakan pada pekerjaan pemrograman dengan mengekspresikannya ke dalam suku-suku dari program lain dengan menambahkan langkah-langkah sejenis. Contoh paling sederhana dari proses rekursi adalah menghitung nilai faktorial dari bilangan bulat. Nilai faktorial, secara rekursif dapat ditulis sebagai :

0! = 1

N! = N x (N-1)!, Untuk N > 0

yang secara notasi pemrograman bisa ditulis sebagai :

FAKTORIAL (0) = 1 1)

FAKTORIAL (N) = N * FAKTORIAL (N-1) 2)

Persamaan 2) di atas merupakan contoh hubungan rekurens (recurrence relation), yang berarti bahwa nilai suatu fungsi dengan argumen tertentu bisa dihitung dari fungsi yang sama dengan argumen yang lebih kecil. Persamaan 1) yang tidak bersifat rekursif, disebut nilai awal. Setiap fungsi rekursi paling sedikit mempunyai 1 (satu) nilai awal; jika tidak, fungsi tersebut tidak bisa dihitung secara eksplisit.

C. Rangkuman

1. Fungsi adalah satu blok kode yang dapat melakukan tugas tertentu atau satu blok instruksi yang dieksekusi ketika dipanggil dari bagian lain dalam suatu program.
2. Tipe fungsi digunakan untuk menentukan tipe keluaran fungsi, Fungsi Tanpa Nilai Balik, Fungsi dengan Nilai Balik, fungsi dengan parameter.

D. Tugas

1. Membuat resume dari materi yang telah diterangkan.
2. Tanya jawab dan diberikan latihan soal terhadap individu.

E. Test

1. Tuliskan sifat-sifat fungsi yang baik.
2. Tuliskan contoh proses rekursif.

MODUL 7
LINKED LIST

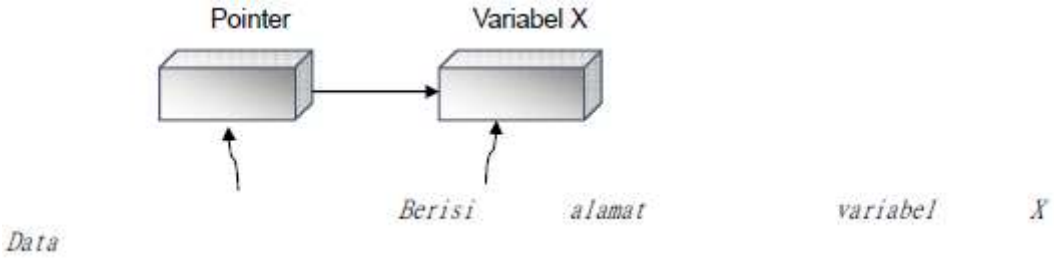
E. Tujuan

Mahasiswa mampu memahami tipe data pointer, deklarasi pointer, variable pointer, variabel dinamis, pointer dan array dinamis, konsep linked list, operasi dasar dalam linked list (senarai berantai), definisi linked list, single linked list, double linked list, representasi simpul (*node*), alokasi simpul, operasi pada linked list, insert sebagai node awal (head) dari linked list, insert setelah node tertentu, insert sebelum node tertentu, circular list.

B. Uraian Materi

Materi 1 : POINTER

Pointer adalah tipe data yang digunakan untuk menunjuk ke suatu data. Suatu variable yang bertipe pointer (variabel pointer) tidak berisi data, melainkan berisi alamat suatu data. Di dalam komputer setiap lokasi data mempunyai alamat yang khas. Gambar berikut contoh suatu pointer yang menunjuk ke suatu data.



Gambar Pointer

Bentuk deklarasi variabel pointer : `tipe *variabel;`

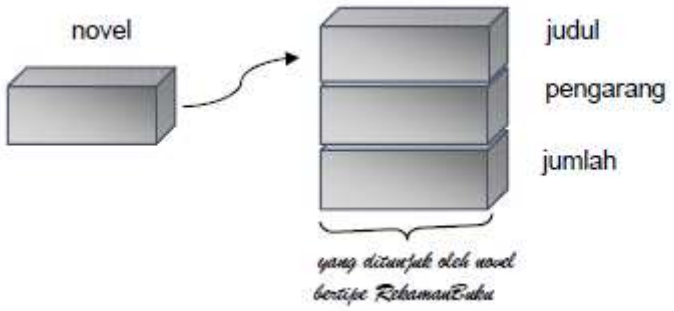
Contoh : `double *p;`

p adalah variabel pointer yang menunjuk ke data bertipe double.

Gambar Mendeklarasikan Pointer

Berdasarkan kondisi di atas, dimungkinkan untuk mengakses data pada variabel X melalui Pointer. Pointer biasa digunakan sehubungan dengan pembentukan variabel dinamis. Variabel Dinamis adalah variabel yang bisa dialokasikan di dalam memori atau dihapus dari memori ketika program dieksekusi.

Variabel Pointer



Gambar Variabel pointer yang menunjuk ke struktur

Supaya suatu variabel pointer menunjuk ke suatu variabel data, penugasan seperti berikut diperlukan :

```
Variabel_pointer = &variabel_data
```

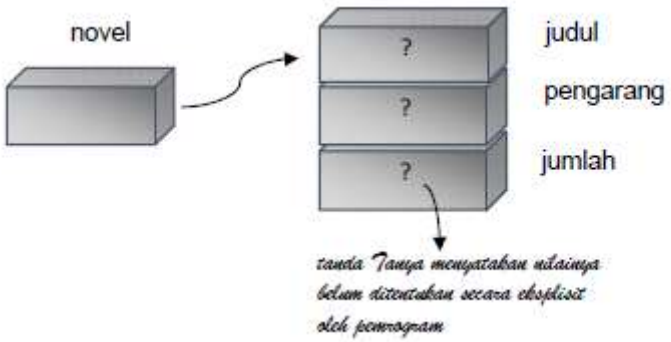
- Simbol & berarti alamat
- Pernyataan di atas berarti bahwa : **variabel_pointer** diisi dengan alamat **variabel_data**.

Variabel Dinamis

Variabel dinamis adalah variabel yang bisa diciptakan ketika program dieksekusi. Menciptakan variabel dinamis butuh variabel pointer, kuncinya yaitu operator **new**. Misalnya terdapat variabel pointer novel yang bertipe pointer. Agar tercipta variabel dinamis yang akan ditunjuk oleh novel, perintahnya adalah :

```
Novel = new RekamanBuku;
```

Gambar berikut menunjukkan keadaan setelah pernyataan tersebut dieksekusi :

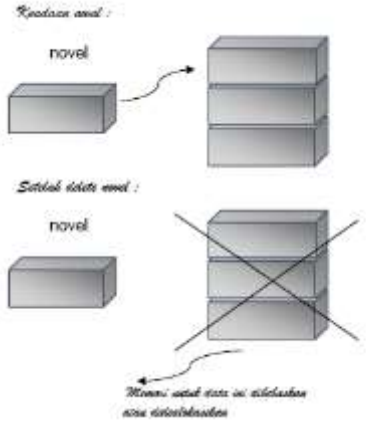


Gambar Keadaan setelah variable dinamis diciptakan

MENGHAPUS VARIABEL DINAMIS

Bila suatu variabel dinamis tidak diperlukan lagi, memori yang digunakannya bisa dihapus. Perintah yang diperlukan untuk keperluan tersebut berupa prosedur delete.
Perintahnya adalah :

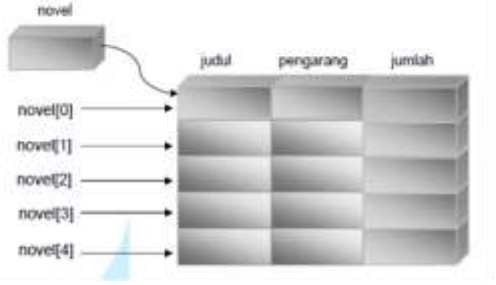
```
Deletevariabel_pointer;
```



Gambar Penghapusan Variabel Dinamis

POINTER dan ARRAY DINAMIS

Array juga dapat dipesan secara dinamis melalui **new**. Hal yang terpenting dalam menggunakan array dinamis adalah penyebutan jumlah elemen array dilakukan dengan menuliskan jumlah elemen dalam tanda [ ].



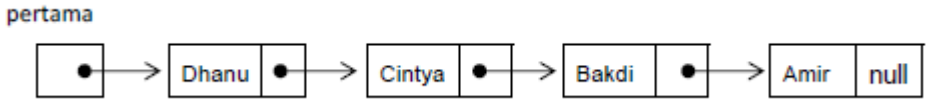
Gambar Pengalokasian Array Dinamis

Materi 2 : KONSEP LINKED LIST

1. Mengenal Struktur Data Linked List

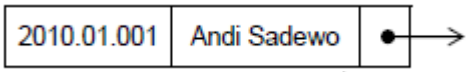
Linked List (Senarai Berantai) adalah jenis struktur data yang berisi kumpulan data yang disusun secara linear dengan setiap data disimpan dalam sebuah simpul dan antara satu simpul dengan simpul lain dihubungkan melalui pointer. Struktur data ini mempunyai bentuk dasar dengan sifat data disisipkan ke dalam senarai melalui salah satu ujungnya.

Contoh linked list :



Perhatikan gambar di atas bahwa Amir sebagai data yang pertama kali dimasukkan akan berada pada salah satu ujung (ujung kanan). Sedangkan data yang terakhir kali dimasukkan yaitu Dhanu, justru berada dibagian kiri, yang ditunjuk oleh pointer pertama. Dengan kata lain, pointer pertama selalu menunjuk ke data yang terakhir kali dimasukkan. Berbeda dengan stack, penghapusan data dalam linked list bias dilakukan di mana saja. Dalam terminologi linked list, setiap data diletakkan dalam sebuah simpul (node). Pada gambar di atas terdapat 4 buah simpul. Setiap simpul terdiri atas 2 bagian, yaitu bagian data dan bagian penunjuk ke simpul berikutnya. Pada contoh di atas bagian data hanya mengandung sebuah data, yaitu nama orang. Namun bagian data bisa saja mengandung beberapa data. Misalnya bagian data terdiri atas nomor mahasiswa dan nama mahasiswa seperti contoh berikut :

Berisi nomor mahasiswa
Dan Nama mahasiswa



- Pointer yang digunakan untuk menunjuk ke simpul berikutnya.
- Berisi NULL kalau tidak menunjuk ke simpul.

2. Operasi Dasar Dalam Linked List (Senarai Berantai)

Operasi dasar pada Linked List berupa :

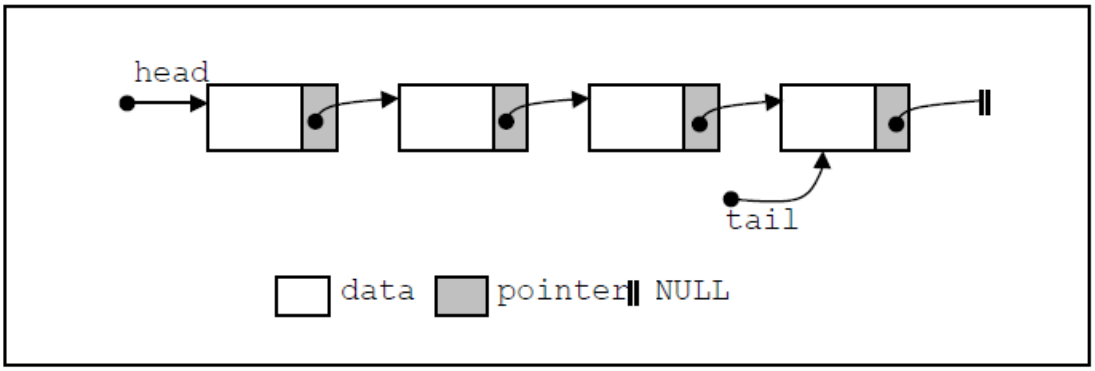
- **INSERT( )**, menyatakan operasi untuk memasukkan data ke dalam linked list pada posisi yang ditunjuk oleh pointer pertama. Operasi ini biasa dinyatakan dengan insert(S, d) atau biasa disingkat insert(d). S menyatakan linked list dan d menyatakan item data yang dimasukkan ke dalam linked list S.
- **FIND( )**, menyatakan operasi untuk mencari suatu data dalam linked list. Operasi ini biasa dinyatakan dengan fungsi **find(pendahulu, x)**. nilai baliknya berupa **true** kalau data yang dicari (yaitu x) ada; atau **false** kalau data yang dicari tidak ada. Pada saat nilai balik bernilai true, pendahulu menunjuk simpul yang berada tepat sebelum simpul yang berisi data yang dicari.
- **REMOVE( )**, menyatakan operasi untuk menghilangkan sebuah simpul dari linked list. Operasi ini biasa dinyatakan dengan remove(S, x) atau remove(x) saja.

Materi 3 : Definisi Linked List

Pengolahan data yang kita lakukan menggunakan komputer seringkali mirip dengan ilustrasi di atas, yang antara lain berupa penyimpanan data dan pengolahan lain dari sekelompok data yang telah terorganisir dalam sebuah urutan tertentu. Salah satu cara untuk menyimpan sekumpulan data yang kita miliki adalah menggunakan larik. Keuntungan dan kerugian pemakaian larik untuk menyimpan sekelompok data yang banyaknya selalu berubah dan tidak diketahui dengan pasti kapan penambahan atau penghapusan akan berakhir.

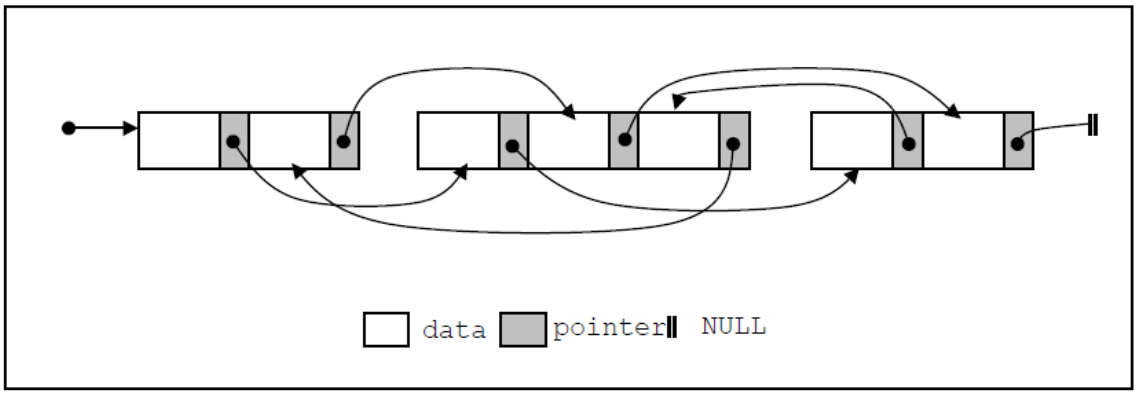
Single Linked List

Single linked list atau biasa disebut linked list terdiri dari elemen-elemen individu, dimana masing masing dihubungkan dengan pointer tunggal. Masing-masing elemen terdiri dari dua bagian, yaitu sebuah data dan sebuah pointer yang disebut dengan pointer *next*. Dengan menggunakan struktur *two-member* seperti ini, linked list dibentuk dengan cara menunjuk pointer *next* suatu elemen ke elemen yang mengikutinya. Pointer *next* pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut *head*, dan elemen terakhir dari suatu list disebut *tail*.



Gambar Elemen yang Dihubungkan Bersama Dalam Bentuk Linked List

Untuk mengakses elemen dalam linked list, dimulai dari head dan menggunakan pointer next dari elemen selanjutnya untuk berpindah dari elemen ke elemen berikutnya sampai elemen yang diminta dicapai. Dengan single linke list, list dapat dilintasi hanya satu arah dari head ke tail karena masing-masing elemen tidak terdapat link dengan elemen sebelumnya. Sehingga, apabila kita mulai dari head dan berpindah ke beberapa elemen dan berharap dapat mengakses elemen sebelumnya, kita harus mulai dari head. Secara konseptual, linked list merupakan deretan elemen yang berdampingan. Akan tetapi, karena elemen-elemen tersebut dialokasikan secara dinamis (menggunakan malloc), sangat penting untuk diingat bahwa kenyataannya, linked list akan terpencarpencar di memori Pointer dari elemen ke elemen berarti sebagai penjamin bahwa semua elemen dapat diakses.



Gambar Elemen Pada Linked List Dihubungkan Secara Terpencar-Pencar pada Alamat Memori

Matri 4 : Representasi Simpul (*Node*)

Struktur node pada linked list merupakan suatu simpul(node) yang berisi pointerke suatu data yang merupakan data dirinya sendiri. Model struktur dari linked list tersebut dalam C adalah sebagai berikut:

```
typedef struct node *list;
struct node {
int datalist;
struct node *next;
};
```

dilanjutkan dengan deklarasi dari pointer ke struktur di atas sebagai berikut:

```
struct node *head;
atau
list head;
```

Alokasi Simpul

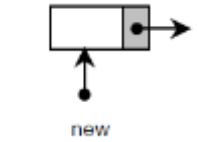
Ketika sebuah variabel dideklarasikan, terlebih dahulu harus diinisialisasi. Demikian juga dengan pengalokasian secara dinamis. Sehingga, fungsi untuk mengalokasikan sebuah node baru, fungsi

allocate_node() menggunakan malloc() untuk mendapatkan memori aktual, yang akan menginisialisasi suatu field data. Next selalu diinisialisasi sebagai NULL. Untuk melihat kemungkinan alokasi memori gagal, maka fungsi allocate_node menghasilkan 0, bila berhasil maka menghasilkan 1. Untuk membebaskan node digunakan fungsi free_node. Fungsi dari alokasi node adalah sebagai berikut :

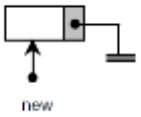
```
int allocate_node(int data, list *new)
{
new = (list) malloc (sizeof(node));
if(new==NULL)
return 0;
new->datalist = data;
new->next=NULL;
return 1;
}
```

10

Untuk inisialisasi list setelah alokasi untuk node pertama maka ditambahkan statement sebagai berikut:
head = new;
Ilustrasi dari fungsi i allocate_node() adalah sebagai berikut :



Untuk inisialisasi list setelah alokasi untuk node pertama ilustrasinya adalah sebagai berikut:
head = new;



Fungsi free_node()menggunakan memori dinamis dengan fungsi free(). free() akan menghancurkan node yang menunjuk ke pointer yang dilewati, sehingga tempat yang ada dapat dipakai untuk lainnya. Akan tetapi, pointer yang melewati free() tidak otomatis menjadi null, tetapi akan menunjuk ke node yang sudah tidak ada. Karena itu pointer harus didefinisikan dengan NULL.

```
void free_node(list p_L)
{
free(p_L);
p_L=NULL;
}
```

Ilustrasi dari fungsi free_node()dapat dilihat pada gambar berikut :

```
free(*p_L);
```

Materi 5 : Operasi pada Linked List

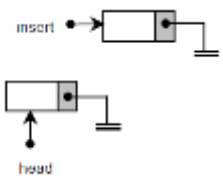
Operasi yang terpenting pada linked list, yaitu menambahkan node (insert) dan menghapus node (delete).
Fungsi insert pada linked list meliputi :
- insert sebagai node awal (head) dari linked list
- insert setelah node tertentu
- insert sebelum node tertentu
- insert sebagai node akhir (tail) dari linked list

Insert sebagai node awal (head) dari linked list

Statement kelanjutan dengan deklarasi seperti diatas untuk insert sebagai node awal (head) dari linked list adalah sebagai berikut:

```
void insertashead(list insert)
{
insert->next=head;
head = insert;
}
```

ilustrasi dari fungsi diatas adalah sebagai berikut:
kondisi awal

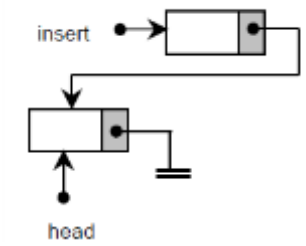


Insert setelah node tertentu

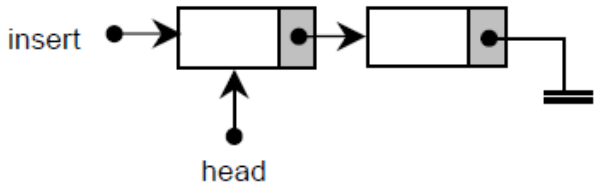
Statement untuk insert setelah node tertentu dari linked list adalah sebagai berikut:

```
void insertafternode(int x, list insert)
{
list after;
after = head;
do
after = after->next;
while (after->datalist != x);
insert->next = after->next;
after->next = insert;
}
```

```
after = after->next;
while (after->datalist != x);
insert->next = after->next;
```




```
after->next = insert;
```



Langkah-langkah untuk proses di atas adalah sebagai berikut:

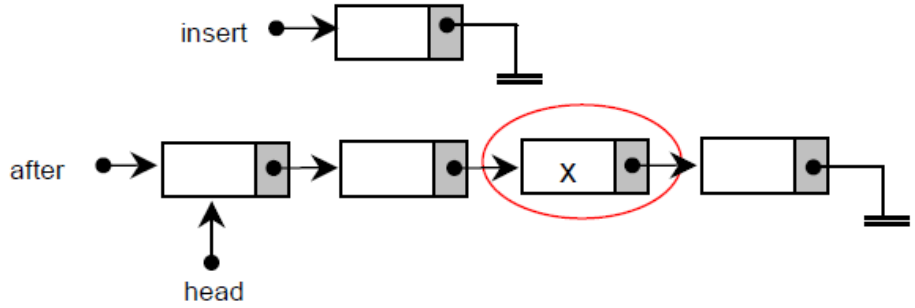
1. Pointer next dari elemen baru menunjuk elemen setelah elemen tertentu
2. Pointer next elemen sebelumnya menunjuk ke elemen baru

Insert sebelum node tertentu

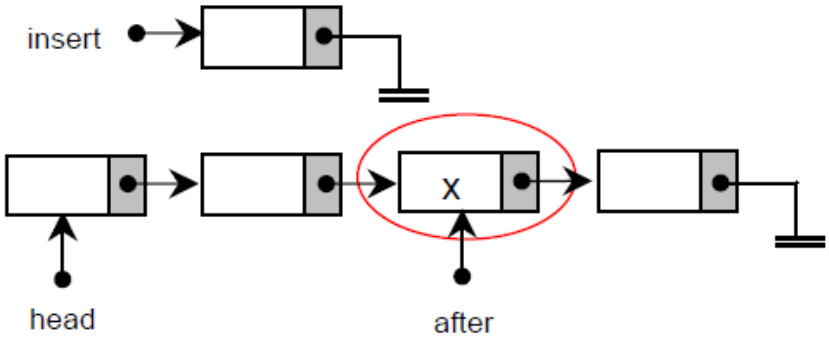
Statement untuk insert setelah node tertentu dari linked list adalah sebagai berikut:

```
void insertbeforenode(int x, list insert)
{
list before, prevbefore;
if (head->datalist = x)
insertashead(insert)
else
{
before = head;
do
prevbefore = before;
before = before->next;
while (before->datalist != x);
insert->next = before;
prevbefore->next = insert;
}
}
```

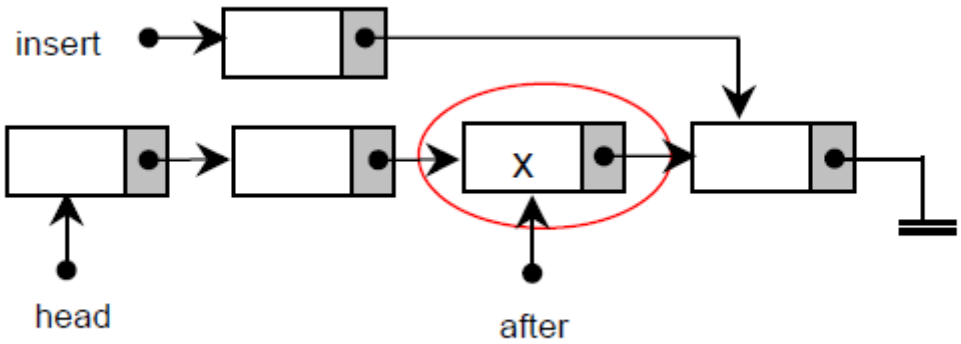
ilustrasi dari fungsi diatas adalah sebagai berikut:
before = head;



do
prevbefore = before;
before = before->next;
while (before->datalist != x);



insert->next = before;



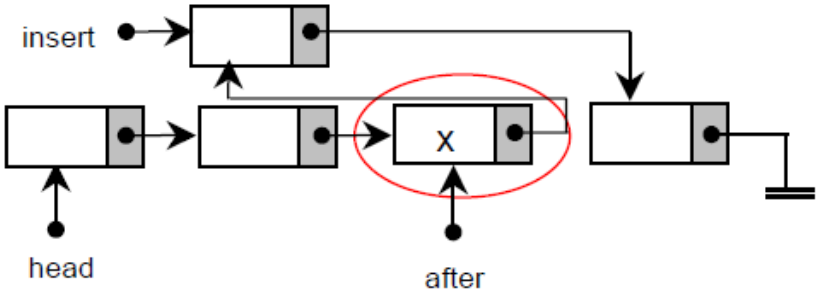
prevbefore->next = insert;

Langkah-langkah untuk proses di atas adalah sebagai berikut:
1. Telusuri list sampai elemen tertentu, catat juga elemen sebelumnya
2. Lakukan penyisipan sebelum elemen tertentu tersebut

Insert di akhir (tail) dari linked list

Statement untuk insert di akhir dari list adalah sebagai berikut:

```
void insertastail(list insert)
{
list tail;
tail = head;
do
```




```
tail->next = insert;
tail = tail->next;
}
```

Langkah-langkah untuk proses di atas adalah sebagai berikut:
1. Telusuri list sampai elemen terakhir (tail->next=NULL)
2. Lakukan pentisipan setelah elemen terakhir

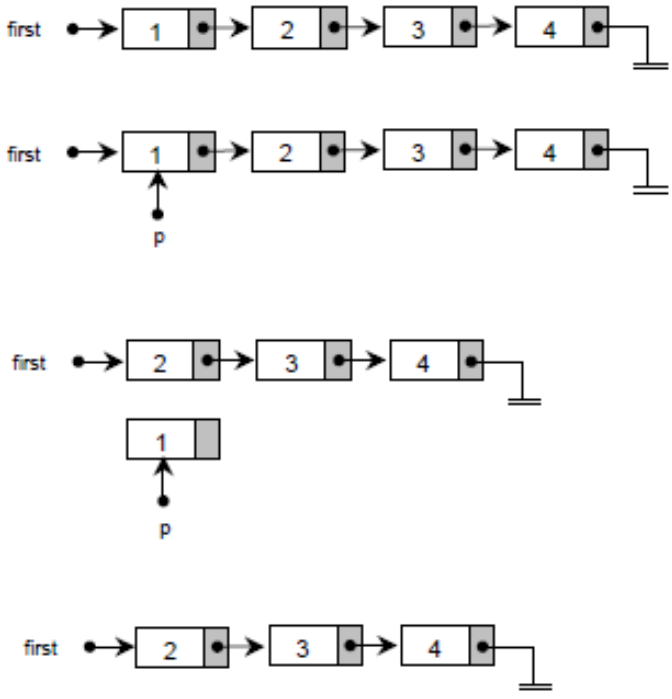
Delete

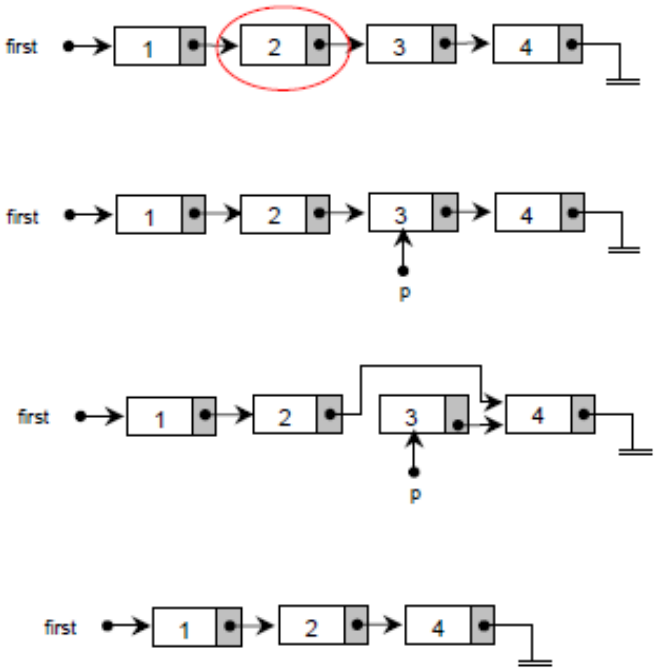
Fungsi delete pada linked list meliputi :
- delete sebagai simpul pertama(head) dari linked list
- delete setelah simpul tertentu
- delete simpul terakhir

Penghapusan Simpul Pertama:

Langkah-langkah untuk proses di atas adalah sebagai berikut:
1. Pointer first diarahkan pada data ke-2
2. Pointer p diarahkan pada data ke-1
3. Bebaskan pointer p (secara otomatis data ke-1 terhapus)

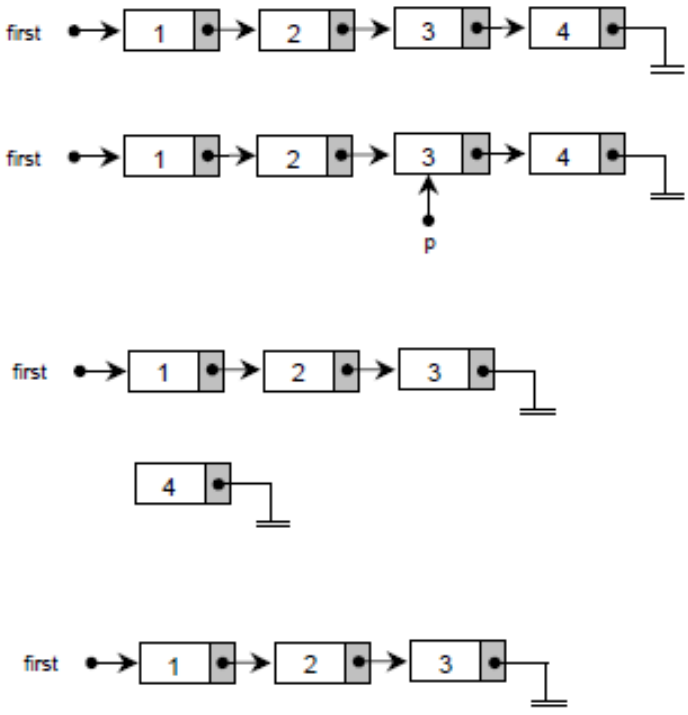
Penghapusan Setelah Simpul Tertentu





Langkah-langkah untuk proses di atas adalah sebagai berikut:
1. Arahkan pointer first s/d data yang ditunjuk
2. Pointer p diarahkan pada first->next
3. Arahkan pointer first->next pada p->next
4. Bebaskan pointer p (secara otomatis data setelah simpul tertentu terhapus)

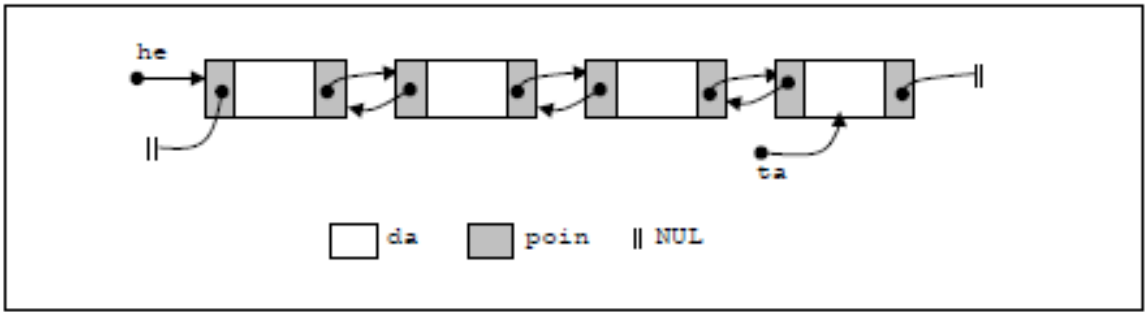
Penghapusan Simpul Terakhir



Langkah-langkah untuk proses di atas adalah sebagai berikut:
1. Telusuri simpul s/d first->next = NULL
2. Arahkan pointer p ke first
3. Bebaskan pointer p->next (Simpul Terakhir)
4. Arahkan p->next ke NULL

Materi 6 : Double Linked List

Elemen-elemen dihubungkan dengan dua pointer dalam satu elemen. Struktur ini menyebabkan list melintas baik ke depan maupun ke belakang. Masing-masing elemen pada double linked list terdiri dari tiga bagian, disamping data dan pointer *next*, masing-masing elemen dilengkapi dengan pointer *prev* yang menunjuk ke elemen sebelumnya. *Double linked list* dibentuk dengan menyusun sejumlah elemen sehingga pointer *next* menunjuk ke elemen yang mengikutinya dan pointer *prev* menunjuk ke elemen yang mendahuluinya. Untuk menunjukkan *head* dari *double linked list*, maka pointer *prev* dari elemen pertama menunjuk NULL. Untuk menunjukkan *tail* dari *double linked list* tersebut, maka pointer *next* dari elemen terakhir menunjuk NULL.

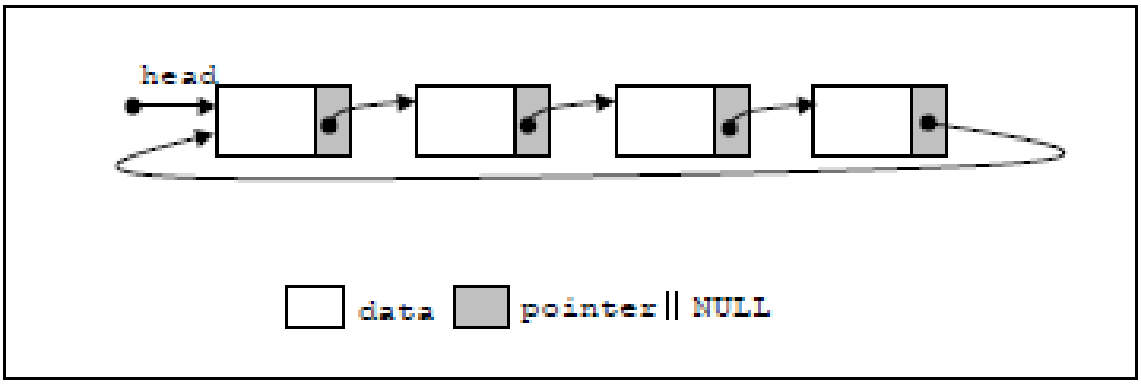


Gambar Elemen yang Dihubungkan dalam Bentuk Double Linked List

Untuk melintas kembali melalui *double linked list*, kita gunakan pointer *prev* dari elemen yang berurutan pada arah *tail* ke *head*. *Double linked list* mempunyai fleksibilitas yang lebih tinggi daripada *single linked list* dalam perpindahan pada list. Bentuk ini sangat berguna ketika akan meletakkan suatu elemen pada list dan dapat memilih dengan lebih bijaksana bagaimana memindahkannya. Sebagai contoh, salah satu fleksibilitas dari *double linked list* adalah dalam hal memindahkan elemen daripada menggunakan *single linked list*.

Materi 7 : Circular List

Circular list adalah bentuk lain dari linked list yang memberikan fleksibilitas dalam melewatkan elemen. Circular list bisa berupa single linked list atau double linked list, tetapi tidak mempunyai tail. Pada circular list, pointer next dari elemen terakhir menunjuk ke elemen pertama dan bukan menunjuk NULL. Pada double linked circular list, pointer prev dari elemen pertama menunjuk ke elemen terakhir. susunan dari single linked circular list, hanya menangani link dari elemen terakhir kembali ke elemen pertama.



Gambar Single Linked Circular List

C. Rangkuman

- Linked list adalah sebuah struktur untuk menyimpan data yang bersifat dinamik
- Beberapa operasi dapat diterapkan pada linked list seperti sisip(insert), hapus(delete)
- Operasi-operasi yang ada pada linked list relatif lebih sulit jika dibandingkan dengan operasi-operasi pada struktur yang statis.

D. Tugas

3. Membuat resume dari materi yang telah diterangkan.
4. Tanya jawab dan diberikan latihan soal terhadap individu.

E. Test

1. Buatlah linked list untuk data mahasiswa, tambahkan prosedur untuk menambah dan menghapus data.
2. Tambahkan tampilan di output pada program di atas untuk menghitung nilai ratarata, nrata=total/jumlah_siswa; total didapatkan dari menambahkan nilai yang didapat tiap mahasiswa.

MODUL 8
STACK, QUEUE

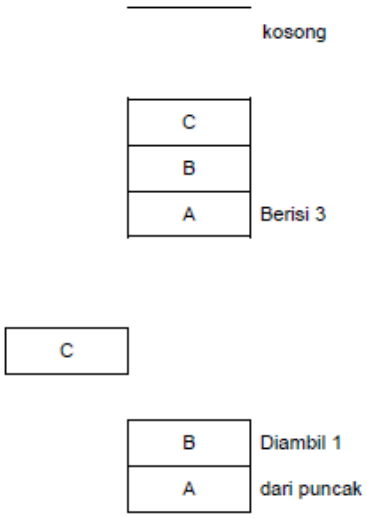
A. Tujuan

Mahasiswa mampu memahami stack (tumpukan), sifat tumpukan Last In First Out (LIFO), operasi dasar PUSH dan POP, operasi-operasi/fungsi stack, aplikasi stack, dasar teori Queue, FIFO (First In First Out), Implementasi Queue dengan Linked List, operasi-operasi standar queue,

B. Uraian Materi

Materi 1 : Pengertian Stack

Stack (tumpukan) adalah struktur data yang memungkinkan penyisipan dan pengambilan data dilakukan dari satu ujung yang disebut puncak. Sesuai namanya, struktur data ini digambarkan seperti keadaan tumpukan piring atau tumpukan buku. Cara yang paling mudah untuk meletakkan piring ke dalam tumpukan yakni dengan menaruhnya dibagian puncak. Begitu juga kalau ingin mengambil piring. Piring diambil dari data yangberada di puncak tumpukan. Penyimpanan data/item dimana data/item yang diakses adalah paling akhir yang disebut top of stack. Item ditempatkan membentuk tumpukan.



Gambar 1. Struktur Data Tumpukan

Tumpukan memiliki sifat Last In First Out (LIFO). Artinya, data yang terakhir kali dimasukkan/disisipkan akan menjadi data yang pertama kali keluar. Pada contoh di atas, yang berisi tumpukan A, B, dan C jelas terlihat bahwa C adalah data yang terakhir kali ditumpukkan. Jika terjadi operasi pengambilan data maka C adalah data yang akan keluar terlebih dulu.

Materi 2 : Operasi-operasi/fungsi Stack :

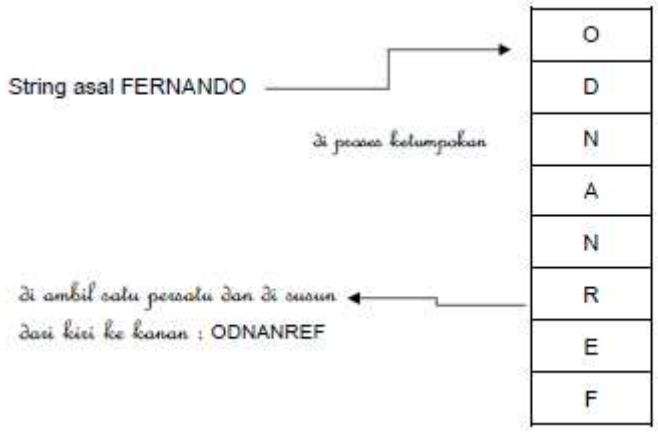
- Push : digunakan untuk menambah item pada stack pada tumpukan paling atas
- Pop : digunakan untuk mengambil item pada stack pada tumpukan paling atas
- Clear : digunakan untuk mengosongkan stack
- IsEmpty : fungsi yang digunakan untuk mengecek apakah stack sudah kosong
- IsFull : fungsi yang digunakan untuk mengecek apakah stack sudah penuh

Operasi dasar pada tumpukan adalah PUSH dan POP.

- PUSH adalah operasi untuk memasukkan data ke dalam tumpukan. Operasi ini biasa dinyatakan dengan push(T,d). T menyatakan tumpukan dan d menyatakan item data yang disisipkan ke dalam tumpukan T.
- POP adalah operasi untuk mengambil data dari tumpukan. Operasi ini biasa dinyatakan dengan pop(T). Dalam hal ini data teratas dari tumpukan T akan dikeluarkan dan menjadi nilai balik pop. Itulah sebabnya, penggunaan pop sering dituangkan dalam bentuk pernyataan : data = pop (T);

Materi 3 : Aplikasi Tumpukan

Aplikasi Tumpukan (Stack) sangat banyak, beberapa penerapan Tumpukan diantaranya adalah m**embalik string.** Jika memproses suatu string dimulai dari yang paling kiri dan menaruh ke tumpukan karakter per karakter maka karakter paling kiri akan berada pada posisi paling bawah dalam tumpukan. Kalau kemudian, karakter dalam tumpukan diambil satu persatu dan disusun dari kiri ke kanan maka string akan terbentuk dengan susunan terbalik terhadap aslinya, seperti diperlihatkan pada gambar di bawah ini.



Mengkonversikan bilangan system decimal ke system biner. Contoh Bilangan 19 identik dengan bilangan biner : 10011. Algoritma untuk melakukannya adalah seperti berikut :

1. Tumpukan <---- kosong
2. While Bilangan > 0
3. Digit <---- sisa pembagian bilangan bulat dengan 2
4. Push(Tumpukan, Digit)
5. Bilangan <---- pembagian bilangan bulat dengan 2
6. End-While
7. While (Tumpukan tidak kosong)
8. Digit <---- Pop
9. Tampilkan digit
10. End-While

Mengevaluasi Ekspresi Aritmetika. Misalnya, tumpukan dipakai untuk memproses perhitungan semacam (2+1)*3+5*2 yang melibatkan berbagai operator dengan prioritas yang berbeda. Memproses pasangan tanda kurung dalam suatu ekspresi. Misalnya, ekspresi seperti (a(b{c|d}[]) dianggap valid, sedangkan ekspresi (a(b{c|d] dianggap tidak valid. Menangani fungsi Rekursif. Secara internal komputer menggunakan tumpukan ketika terjadi pemanggilan fungsi secara rekursif.

Materi 4 : Queue

Queue disebut juga antrian dimana data masuk di satu sisi dan keluar di sisi yang lain. Karena itu, queue bersifat FIFO (First In First Out). Antrian (*Queue*) merupakan suatu kumpulan data yang penambahan elemennya (masuk antrian) hanya bisa dilakukan pada suatu ujung (disebut dengan sisi belakang/*rear*) atau disebut juga *enqueue* yaitu apabila seseorang masuk ke dalam sebuah antrian. Jika seseorang keluar dari antrian/penghapusan (pengambilan elemen) dilakukan lewat ujung yang lain (disebut dengan sisi depan/*front*) atau disebut juga *dequeue* yaitu apabila seseorang keluar dari antrian. Jadi, dalam antrian menggunakan prinsip "masuk pertama keluar pertama" atau disebut juga dengan prinsip FIFO (*first in first out*). Dengan kata lain, urutan keluar akan sama dengan urutan masuknya. Contoh : antrian mobil saat membeli karcis di pintu jalan tol, antrian di bioskop dan sebagainya.

Dasar Teori

Queue atau antrian adalah suatu kumpulan data yang penambahan elemennya hanya bisa dilakukan pada suatu ujung (disebut dengan sisi belakang atau rear), dan penghapusan atau pengambilan elemen dilakukan lewat ujung yang lain (disebut dengan sisi depan atau front). Kalau tumpukan dikenal dengan menggunakan prinsip LIFO (Last In First Out), maka pada antrian prinsip yang digunakan adalah FIFO (First In First Out).

Operasi / Prosedur Standar pada QUEUE / ANTRIAN

QUEUE merupakan struktur data dinamis, ketika program dijalankan, jumlah elemennya dapat berubah secara dinamis sesuai keperluan. Berikut ini operasi-operasi standar pada queue :

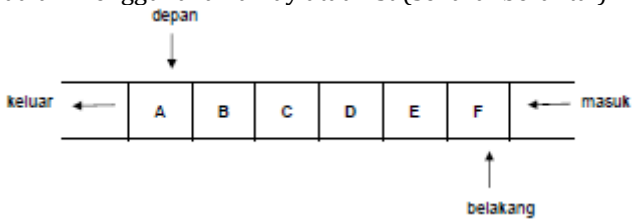
a. **Inisialisasi**, merupakan prosedur untuk membuat queue pada kondisi awal, yaitu queue yang masih kosong (belum mempunyai elemen).
b. **InQueue**, Insert Queue merupakan prosedur untuk memasukkan sebuah elemen baru pada queue. Jumlah elemen Queue akan bertambah satu dan elemen tersebut merupakan elemen belakang.
c. **DeQueue**, Delete Queue merupakan prosedur untuk menghapus/mengambil sebuah elemen dari queue. Elemen yang diambil adalah elemen depan dan jumlah elemen queue akan berkurang satu.

Operasi-operasi yang berhubungan dengan jumlah elemen suatu queue adalah :

1. ***Size***, yaitu operasi untuk mendapatkan banyaknya elemen queue.
2. ***Empty***, yaitu prosedur untuk mengetahui apakah queue dalam keadaan kosong atau tidak. Dengan status ini maka dapat dicegah dilakukannya operasi Dequeue dari suatu queue yang kosong.
3. ***Full***, merupakan prosedur untuk mengetahui apakah Queue penuh atau tidak.
   Prosedur ini hanya berlaku untuk queue yang jumlahnya terbatas.

Materi 5 : Implementasi Queue dengan Array Linear dan Sirkular

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



Gambar di atas menunjukkan contoh penyajian antrian menggunakan array. Antrian di atas berisi 6 elemen, yaitu A, B, C, D, E dan F. Elemen A terletak di bagian depan antrian dan elemen F terletak di bagian belakang antrian. Jika ada elemen baru yang akan masuk, maka elemen tersebut akan diletakkan di sebelah kanan F. Dan jika ada elemen yang akan dihapus, maka A akan dihapus terlebih dahulu. Elemen A terletak di bagian depan, kemudian disusul elemen B dan elemen yang paling akhir atau paling belakang adalah elemen F. Misalkan ada elemen baru yang akan masuk maka akan terletak di belakang elemen F, sehingga elemen baru akan menempati posisi yang paling belakang.

Seperti pada tumpukan atau stack di dalam antrian juga dikenal 2 operasi dasar yaitu menambah elemen baru yang akan diletakkan di bagian belakang antrian dan menghapus elemen yang terletak di bagian depan antrian.

Untuk memahami penggunaan antrian dalam array, kita membutuhkan deklarasi
antrian, misalnya sebagai berikut :

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

Dengan deklarasi di atas, elemen antrian dinyatakan dalam tipe integer yang semuanya terdapat dalam struktur. Variabel first menunjukkan posisi elemen pertama dalam array, dan variabel last menunjukkan posisi elemen terakhir dalam array.
Algoritma dari penggalan program di atas adalah :
1. Tentukan elemen yang akan dimasukkan ke dalam antrian (dalam hal ini adalah 6 elemen).
2. Deklarasikan struktur untuk menampung elemen pada antrian.
3. Selesai
Untuk menambah elemen baru dan mengambil elemen dari antrian dalam antrian, diperlukan deklarasi berikut ini :

```
void initialize_queue ( Tqueue *Pqueue)
{
Pqueue -> firs = 0;
Pqueue -> last = -1;
Pqueue -> number_of_items = 0;
}
Tboolean enqueue ( Tqueue *Pqueue, Titem item)
{
if (Pqueue -> number_of_items >= MAXN)
return (NOT_OK);
else {
Pqueue -> last++;
if (Pqueue -> last > MAXN -1)
Pqueue -> = 0;
Pqueue -> array[Pqueue->last] = item;
Pqueue -> number_of_items++;
return (OK);
}
}
Tboolean dequeue (Tqueue *Pqueue, Titem, item)
{
if (Pqueue -> number_of_items == 0)
return (NOT_OK);
else {
*Pitem = Pqueue -> array[Pqueue->first++];
if (Pqueue -> first > MAXN -1)
Pqueue -> first = 0;
Pqueue -> number_of_items--;
return (OK);
}
}
```

Materi 6 : Implementasi Queue dengan Linked List
Untuk memanipulasi sebuah antrian bisa digunakan dua buah variabel yang menyimpan posisi elemen paling depan dan elemen paling belakang. Apabila antrian diimplementasikan menggunakan linked list maka cukup 2 pointer yang bisa dipakai yaitu elemen paling depan (kepala) dan elemen paling belakang (ekor).
Untuk mengimplementasikan antrian dengan menggunakan pointer, algoritma berikut ini :
1. Tentukan struktur untuk menampung node yang akan dimasukkan pada antrian.
Deklarasi struktur pada penggalan program berikut ini :

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

2. Deklarasikan penambahan elemen baru pada antrian, dimana letaknya adalah paling belakang. Deklarasi penambahan elemen baru tersebut dapat dilihat pada penggalan program berikut ini :

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

```
else
( *tailPtr ) -> nextPtr = newPtr;
*tailPtr = newPtr;
}
```

3. Lakukan pengecekan terhadap antrian, apakah antrian dalam kondisi kosong atau tidak. Kalau kondisi antrian kosong, maka elemen bisa dihapus. Penggalan program berikut ini akan menunjukkan kondisi tersebut.

```
int isEmpty ( QUEUENODEPTR headPtr )
{
return headPtr == NULL;
```

C. Rangkuman

- Stack (tumpukan) adalah struktur data yang memungkinkan penyisipan dan pengambilan data dilakukan dari satu ujung yang disebut puncak.
- QUEUE merupakan struktur data dinamis, ketika program dijalankan, jumlah elemennya dapat berubah.
- Tumpukan dikenal dengan menggunakan prinsip LIFO (Last In First Out), maka pada antrian prinsip yang digunakan adalah FIFO (First In First Out).

D. Tugas

1. Membuat resume dari materi yang telah diterangkan.
2. Tanya jawab dan diberikan latihan soal terhadap individu.

MODUL 9
LINKED LIST

F. Tujuan

Mahasiswa mampu memahami struktur data hirarkis tree, Cara Penggambaran Tree, Terminologi tree, karakteristik tree, karakteristik binary tree, Complete Binary Tree, Full Binary Tree, Preorder, inorder, postorder, traversal binary tree, implementasi binary tree, AVL tree, heap tree, Struktur Data, Konektivitas pada Undigraph, jenis graph, Graph berbobot (weighted graph) Graph , Algoritma Depth First Search, Depth First Search (DFS) :

B. Uraian Materi

Materi 1 : Struktur Data Hirarkis : Tree

Tree merupakan salah satu struktur data yang paling penting, karena banyak aplikasi menggunakan informasi dan data yang secara alami memiliki struktur hirarkis berguna dalam membantu memecahkan banyak masalah algoritmis.

Aplikasi pohon biner :

a. sebagai representasi ekspressi matematika

b. aplikasi pohon biner dalam Huffman Coding.

Cara Penggambaran Tree :

- Notasi Kurung
- Diagram Venn
- Notasi Tingkat
- Notasi Garis

Terminologi :

- **Tree (atau pohon)** sejumlah node yang berhubungan secara hirarkis dimana suatu node pada suatu hirarki merupakan cabang dari node dengan hirarki yang lebih tinggi dan juga memiliki cabang ke beberapa node lainnya dengan hirarki yang lebih rendah.
- **Root (atau akar)**
  Node dengan hirarki tertinggi dinamakan root.
- **leaf (atau daun)**
  node yang tidak memiliki cabang.
- **Internal node (atau node dalam)**
  node yang bukan merupakan leaf.
- **edge (atau sisi atau cabang)**
  menyatakan hubungan hirarkis antara kedua node yang terhubungkan, biasanya digambarkan berarah (berupa panah) untuk menunjukkan node asal edge lebih tinggi dari node tujuan dari edge.
  - Level (atau tingkat) suatu node (Kadang dimulai level 0 atau 1)
  - Bilangan yang menunjukan hirarki dari suatu node, root memiliki level 1, node cabang dari root memiliki level 2, node cabang berikutnya dari node adalah level 3, dan seterusnya.
  - Height (atau tinggi) suatu tree, sama dengan level dengan angka terbesar (hirarki terendah) suatu node yang ada dalam tree atau bisa juga didefinisikan sebagai jumlah sisi terbanyak dari root hingga suatu leaf yang ada di tree.
  - Depth (atau kedalaman) suatu node : jumlah sisi dari root hingga node ybs.
  - Subtree (atau subpohon), sebagian dari tree mulai dari suatu node N melingkupi node-node yang berada dibawah hirarkinya sehingga dapat dipandang sebagai suatu tree juga yang mana N sebagai root dari tree ini.
  - Tree kosong : suatu tree yang tidak memiliki satu node pun.
  - Tree dengan urutan letak geometris node-node yang merupakan cabang yang sama dari suatu node adalah penting; biasanya urutan dari kiri ke kanan. B, C, D dan E memiliki induk (parent) yang sama yaitu A, sehingga ke- 4 node tersebut disebut saudara (siblings)

- Semua node di bawah suatu node induk hanya dapat diakses melalui induknya
- Lintasan (path) sebuah node X adalah urutan akses untuk mendapatkan X yang dimulai dari Akar. Path(M) = [A – B – G – M]
- Panjang lintasan (path length) X adalah jumlah node yang harus diakses untuk mendapatkan X; sehingga |Path(M)| = 4. Ada juga yang menyatakan panjang lintasan sebuah node adalah jumlah garis dari akar sampai node tersebut.
- Tinggi (height) dari sebuah pohon adalah panjang lintasan terpanjang.
  Tinggi pohon di bawah ini = 4
- jika ada sebuah lintasan dari node a ke node b, maka a adalah pendahulu (ancestor) dari b dan b disebut keturunan (descendent) dari a
- Kedalaman (depth) dari sebuah node adalah panjang lintasan dari akar ke node tersebut. Depth (G) = 3

Karakteristik :

a. Terdapat 1 node yang unik, yang tidak memiliki predecessor, yang disebut dengan root (akar)
b. Terdapat satu atau beberapa node yang tidak memiliki successor yang disebut dengan leaf (daun)
c. Setiap node kecuali root pasti memiliki satu predecessor
d. Setiap node kecuali leaf pasti memiliki 1 atau lebih successor

Materi 2 : BINARY TREE

Karakteristik : Maksimum child adalah 2 (Left Child dan Right Child)

Complete Binary Tree : Bila semua node kecuali Leaf memiliki 0 atau 2 child. Subtree pada Heap Tree dapat memiliki path length yang berbeda Skewed Binary Tree (Miring) : Bila semua node, kecuali Leaf memiliki hanya 1 child

Full Binary Tree : Bila semua node kecuali Leaf memiliki 2 Child dan semua subtree harus memiliki path yang sama

Representasi :

- Ekspresi MM dengan Binary Tree

Huffman Code:

Hubungan Level dengan Jumlah Data :

Level Ke :	Jumlah data	
	Minimum	Maksimum
1	1	$1 = 2^0$
2	1	$2 = 2^1$
3	1	$4 = 2^2$
4	1	$8 = 2^3$
5	1	$16 = 2^4$
6	1	$32 = 2^5$
...	...	...
L	1	$2^{(L-1)}$

Ekspressi Matematika (EM) didefenisikan sebagai berikut:

(1). Variabel dan Bilangan adalah EM
(2). Jika E, E1 dan E2 adalah EM maka
– E, + E dan (E) adalah EM
E1 + E2, E1 – E2, E1 * E2, E1 / E2, E1 div E2, E1 mod E2 dan
E1^E2 juga adalah EM
Contoh :
4 adalah EM
X adalah EM
4 + X adalah EM
2 + 4 * 3 adalah EM

(2 + 4) * 3 adalah EM
((3 + 2) * (5 – 2)) / (2 + 3) adalah EM dan seterusnya
Contoh:
Diketahui: Ekspressi Matematika ((2 + 3) * (5 – 2) + 5) * (5 + 3 * 2)
Ditanya: Gambarkan pohon ekspressi dengan cara Bootom Up
Penyelesaian:
Ekspressi tersebut terbagi dua menjadi E1 dan E2 oleh operator *; sehingga ekspressi tersebut dapat ditulis sebagai (E1) * (E2) dimana E1 = ((2 + 3) * (5 – 2) + 5) dan E2 = (5 + 3 * 2)

Materi 3 : TRAVERSAL BINARY TREE

- Operasi penelusuran node-node dalam binary tree
- Traversal dibagi 3, yaitu :

a. Preorder
b. Inorder
c. Postorder

Implementasi Binary Tree :

a. Array
b. Linked List

Implementasi BT pada array :

- Indeks pada array menyatakan nomor kode
- Node root mempunyai indeks array = 1
- Leftchild suatu node dengan nomor p adalah (2p)
- Rightchild suatu node dengan nomor p adalah (2p+1)
- Parent dari suatu node dengan nomor p adalah (p div 2)

Implementasi Binary Search Tree :

Ketentuan :

a. Semua left child harus lebih kecil dari parent
b. Semua right child harus lebih besar dari parent

Keuntungan : Pencarian node target menjadi lebih efisien dan cepat

Operasi-Operasi Standar BST :

- Mengosongkan BST
- Mencek apakah BST kosong
- Mencari Tree Minimum
- Mencari Tree Maksimum
- Memasukkan data baru ke dalam BST
- Mencari elemen tertentu dalam BST
- Menghapus data tertentu dari dalam BST
- Menampilkan semua elemen dalam BST

AVL TREE

Definisi : BST yang mempunyai ketentuan bahwa maks perbedaan tinggi antara subtree kiri dan kanan adalah satu Height-Balanced Tree :

- BST adalah Height Balanced p-tree yang artinya maksimum perbedaan height antara subtree kiri dan kanan adalah p
- AVL Tree adalah height balanced 1-tree yang berarti maksimum perbedaan height antara subtree kiri dan kanan adalah satu

a. TallLeft bila subtree kiri lebih panjang dari subtree kanan (simbol -)
b. TallRight bila subtree kanan lebih panjang dari subtree kiri (simbol +)
c. Balance bila Subtree kiri dan kanan mempunyai height
d. sama, simbolnya 0

Search Path :

Path pencarian lokasi untuk dilakukan operasi insert dimulai dari Root Pivot Point :
Adanya node pada search path yang balancenya TallLeft (tanda -) atau TallRight (tanda +) dan terletak paling dekat dengan node yang baru, Contoh : Jika diinsert node baru dengan nilai 5, maka pivot pointnya di node 25
Operasi Insert :

- Kasus-1 Tidak ada pivot point dan setiap node adalah balance, maka bisa langsung diinsert sama seperti BST
- Kasus 2
  Jika ada PP tetapi subtree yang akan ditambahkan node baru memiliki height yang lebih kecil, maka bisa langsung di insert
- Kasus 3
  Jika ada PP dan subtree yang akan ditambahkan node baru memiliki height yang lebih besar, maka tree harus degenerate supaya tetap menghasilkan AVL Tree

Regenerate :

- Single Rotation
  a. Single Left Rotation
  b. Single Right Rotation
- Double Rotation
  a. Double Left Rotation
  b. Double Right Rotation

HEAP TREE :

Definisi :

- Heap adalah Tree yang memenuhi persamaan :
  R[I] < R[2*I] dan R[I] < R[2*I + 1]
- Heap disebut juga Complete Binary Tree
- Minimum Heap : Jika nilai parentnya selalu lebih kecil dari pada kedua childrennya
- Maximum Heap : Jika nilai parentnya selalu lebih besar dari pada kedua Childrennya

B-Tree :

- Definisi : Tree yang setiap nodenya dapat berisi lebih dari pada satu elemen
- Jumlah elemen dalam sebuah node tergantung kepada order dari B-Tree tersebut Jumlah minimum elemen dalam setiap node (kecuali root) adalah d, dan jumlah max adalah 2d, dimana d adalah order. Untuk Root, jumlah minimum elemen 1, dan jumlah maksimum adalah 2d
- Jumlah minimum children dari suatu node dalam B-Tree adalah 0, dan
- jumlah Max-nya adalah jumlah elemen+1

Materi 4 : B-Tree

- Definisi :
- Tree yang setiap nodenya dapat berisi lebih dari pada satu elemen Jumlah elemen dalam sebuah node tergantung kepada order dari B-Tree tersebut.
- Jumlah minimum elemen dalam setiap node (kecuali root) adalah d, dan jumlah max adalah 2d, dimana d adalah order. Untuk Root, jumlah minimum elemen 1, dan jumlah maksimum adalah 2d.
- Jumlah minimum children dari suatu node dalam B-Tree adalah 0, dan jumlah Max-nya adalah jumlah elemen+1.

Operasi dalam B-Tree :

- Insert : Jika node belum penuh (jumlah elemen < 2d), maka elemen bisa langsung di-insert
- Delete :

- Jika target node yang akan dihapus berisi elemen lebih dari d, maka target elemen bisa langsung dihapus tanpa perlu
- Jika target node yang akan dihapus berisi d node, penghapusan langsung akan menyebabkan underflow
- Regenerate dilakukan dengan meminjam elemen yang berada di node kiri atau dikanan (yang memiliki elemen lebih dari d). Parent/ Separator akan berubah
- Jika node dikiri atau dikanan yang akan dilakukan peminjaman ternyata mempunyai elemen kurang dari d, yang mana jika dilakukan peminjaman, node tersebut akan terjadi underflow
- Regenerate akan dilakukan dengan menggabungkan node yang akan dihapus dengan node dikiri/kanan

Materi 5 : Graph

Struktur Data Graph :

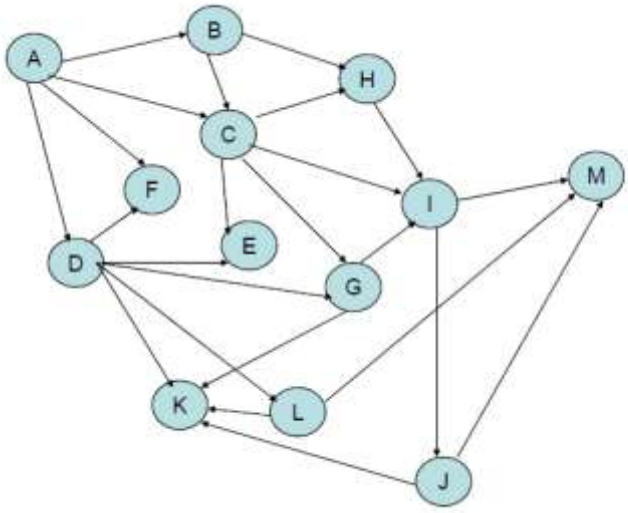
- Graph : struktur data yang berbentuk network/jaringan, hubungan antar elemen adalah many-to-many
- Struktur Data Linear = keterhubungan sekuensial antara entitas data
- Struktur Data Tree = keterhubungan hirarkis
- Struktur Data Graph = keterhubungan tak terbatas antara entitas data.
  Contoh graph : Informasi topologi jaringan dan keterhubungan antar kota-kota
- Keterhubungan dan jarak tidak langsung antara dua kota = data keterhubungan langsung dari kota-kota lainnya yang memperantarainya.
- Penerapan struktur data linear atau hirarkis pada masalah graph dapat dilakukan tetapi kurang efisien. Struktur data graph secara eksplisit menyatakan keterhubungan ini sehingga pencariannya langsung (straightforward) dilakukan pada strukturnya sendiri.

Graph terdiri dari himpunan verteks (node) dan himpunan sisi (edge, arc).
Verteks menyatakan entitas-entitas data dan sisi menyatakan
keterhubungan antara verteks.

Notasi matematis graph G adalah :
G = (V, E)
Sebuah sisi antara verteks *x* dan *y* ditulis {*x*, *y*}.
Subgraph : graph yang merupakan suatu subset (bagian) graph yang connected
Graph H = (*V1*, *E1*) disebut subgraph dari graph G jika *V1* adalah himpunan bagian dari *V* dan *E1* himpunan bagian dari *E*.

Materi 6 : Jenis Graph

a. Directed Graph (Digraph) : jika sisi-sisi graph hanya berlaku satu arah.
   Misalnya : {x,y} yaitu arah *x* ke *y, bukan* dari *y* ke *x*; *x* disebut origin dan *y* disebut terminus. Secara notasi sisi digraph ditulis sebagai vektor (*x*, *y*).
   Contoh Digraph G = {V, E} :
   V = {A, B, C, D, E, F, G, H, I,J, K, L, M}
   E = {(A,B),(A,C), (A,D), (A,F), (B,C), (B,H), (C,E), (C,G), (C,H), (C,I), (D,E), (D,F), (D,G), (D,K), (D,L), (E,F), (G,I), (G,K), (H,I), (I,J), (I,M), (J,K), (J,M), (L,K), (L,M)}.

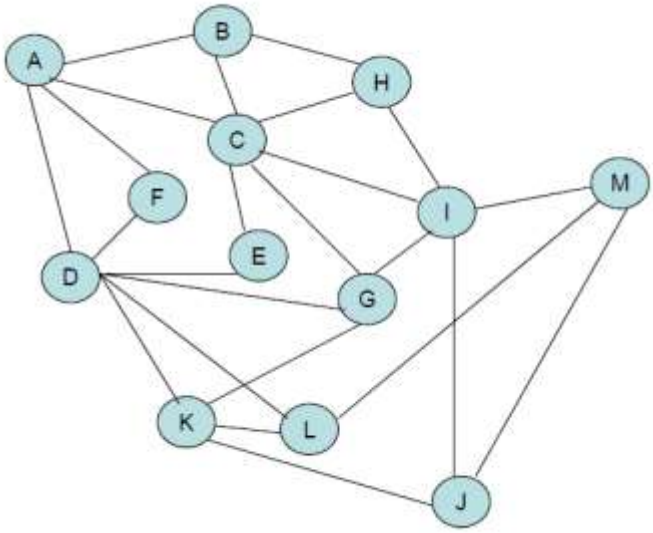


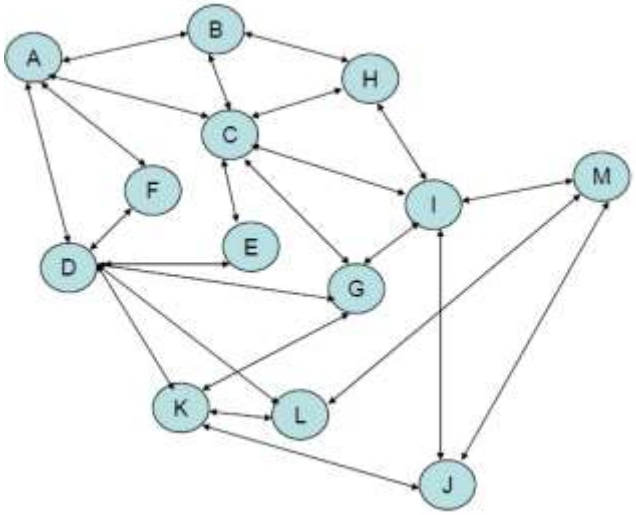
b. Graph Tak Berarah (undirected graph atau undigraph): setiap sisi {*x*, *y*} berlaku pada kedua arah: baik *x* ke *y* maupun *y* ke *x*. Secara grafis sisi pada undigraph tidak memiliki mata panah dan secara notasional menggunakan kurung kurawal.
   Contoh Undigraph G = {V, E}
   V = {A, B, C, D, E, F, G, H, I,J, K, L, M}
   E = { {A,B},{A,C}, {A,D}, {A,F}, {B,C}, {B,H}, {C,E}, {C,G}, {C,H}, {C,I}, {D,E},
   {D,F}, {D,G}, {D,K}, {D,L}, {E,F}, {G,I}, {G,K}, {H,I}, {I,J}, {I,M}, {J,K},
   {J,M}, {L,K}, {L,M}}.

Khusus graph, undigraph bisa sebagai digraph (panah di kedua ujung edge berlawanan)

Struktur data linear maupun hirarkis adalah juga graph. Node-node pada struktur linear ataupun hirarkis adalah verteks-verteks dalam pengertian graph dengan sisi-sisinya menyusun node-node tersebut secara linear atau hirarkis. Struktur data linear adalah juga tree dengan pencabangan pada setiap node hanya satu atau tidak ada.

Linear 1-way linked list (digraph), linear 2-way linked list (undigraph).





Konektivitas pada Undigraph,

- Adjacency: Dua verteks *x* dan *y* yang berlainan disebut terhubung langsung (adjacent) jika ada sisi {*x*, *y*} dalam E.
- Path : Urutan dari kumpulan node-node, dimana tiap node dengan node berikutnya dihubungkan dengan Edge
- Simple Path : Jika node dalam path tsb hanya muncul 1 kali
- Panjang dari path: jumlah sisi yang dilalui path.
- Siklus (cycle) : suatu path dengan panjang lebih dari satu, dimana vertex awal dan akhir sama.
- Siklus sederhana: dalam undigraph, siklus yang terbentuk dari tiga atau lebih verteks yang berlainan, dimana tidak ada vertex muncul lebih satu kali kecuali verteks awal/akhir.
- Dua verteks *x* dan *y* yang berbeda dalam suatu undigraph disebut berkoneksi (connected) apabila ada path penghubung.
- Suatu komponen terkoneksi (connected components) adalah subgraph (bagian dari graph) yang berisikan satu himpunan bagian verteks yang berkoneksi.

Konektivitas pada Digraph

Terminologi pada Undigrap berlaku, kecuali dalam digraph harus dikaitkan dengan arah tertentu karena pada arah yang sebaliknya belum tentu terdefinisi.

Adjacency ke/dari : Jika ada sisi (*x*,*y*) maka pada digraph dikatakan bahwa *x* "adjacent ke" *y* atau *y* "adjacent dari" *x*. Demikian pula jika terdapat path dari *x* ke *y* maka belum tentu ada path dari *y* ke *x*. Jadi dalam digraph keterkoneksian didefinisikan lebih lanjut lagi sebagai berikut.

- Terkoneksi Kuat : Bila setiap pasangan verteks berbeda *x* dan *y* dalam S, *x* berkoneksi dengan *y* dan *y* berkoneksi dengan *x*
- Terkoneksi Lemah : Bila setiap pasangan verteks berbeda *x* dan dalam S, salah satu: *x* berkoneksi dengan *y* (atau *y* berkoneksi dengan *x*) dan tidak kebalikan arahnya (hanya terdefinisi satu path dari *x* ke *y* atau sebaliknya dari *y* ke *x*).

Graph berbobot (weighted graph)

Graph dengan sisi mempunyai Bobot/ Biaya. “Biaya" ini bisa mewakili banyak aspek: biaya ekonomi suatu aktifitas, jarak geografis/tempuh, waktu tempuh, tingkat kesulitan, dan lain sebagainya.

Contoh :
Graph ini merupakan Undirected Weighted Graph. Order dari verteks A = 4,
verteks B = 3, dst. Adjacentcy list dari D adalah = {A, E, F, G, K, L}.

Representasi Graph

Representasi Matriks Keterhubungan Langsung (Adjacency Matrix) Matriks digunakan untuk himpunan adjacency setiap verteks. Baris berisi vertex asal adjacency, sedangkan kolom berisi vertex tujuan adjacency. Bila sisi graph tidak mempunyai bobot, maka [x, y] adjacency disimbolkan dengan 1 dan 0 bila tidak adjacency. Bila sisi graph mempunyai bobot, maka [x, y] adjacency disimbolkan dengan bobot sisi tersebut, dan bila tidak disimbolka ¥.

Materi 7 : Algoritma-algoritma Pencarian

Pencarian vertex adalah proses umum dalam graph. Terdapat 2 metoda pencarian:
Depth First Search (DFS): pada setiap pencabangan, penelusuran verteks-verteks yang belum dikunjungi dilakukan secara lengkap pada pencabangan pertama, kemudian selengkapnya pada pencabangan kedua, dan seterusnya secara rekursif.
Breadth First Search (BFS): pada setiap pencabangan
penelusuran verteks-verteks yang belum dikunjungi dilakukan pada verteks-verteks adjacent, kemudian berturut-turut selengkapnya pada masing-masing pencabangan dari setiap verteks adjacent tersebut secara rekursif.

Algoritma Depth First Search

Algoritma diawali pada vertex S dalam G Kemudian algoritma menelusuri graph dengan suatu insiden edge (*u*,*v*) ke current vertex *u*. Jika edge (*u*, *v*) menunjuk ke suatu vertex v yang siap untuk dikunjungi, maka kita ikuti jalur mundur ke current vertex u. Jika pada sisi lain, edge (u, *v*) menunjuk ke vertex *v* yang tidak dikunjungi, maka kita pergi ke *v* dan *v* menjadi current vertex.
di proses ini hingga kita mencapai sasaran akhir. Pada titik ini, kita mulai jalur mundur. Proses ini berakhir ketika jalur mundur menunjuk balik ke awal vertex.

Algoritma Breadth First Search :

BFS diawali dengan vertex yang diberikan, yang mana di level 0. Dalam stage pertama, kita kunjungi semua vertex di level 1. Stage kedua, kita kunjungi semua vertex di level 2. Disini vertex baru, yang mana adjacent ke vertex level 1, dan seterusnya. Penelusuran BFS berakhir ketika setiap
vertex selesai ditemui.

Implementasi algoritma BFS

Algoritma BFS menjadi kurang straightforward jika dinyatakan secara rekursif. Jadi sebaiknya diimplementasikan secara nonrekursif dengan memanfaatkan queue sebagai struktur data pendukung

Masalah Pencarian Lintasan Terpendek

Pencarian shortest path (lintasan terpendek) adalah masalah umum dalam suatu weighted, connected graph. Misal : Pencarian jaringan jalan raya yang menghubungkan kota-kota disuatu wilayah Lintasan terpendek yag menghubungkan antara dua kota berlainan tertentu (Single-source Single-destination Shortest Path Problems), Semua lintasan terpendek masing-masing dari suatu kota ke setiap kota lainnya (Single-source Shortest Path problems). Semua lintasan terpendek masing-masing antara tiap kemungkinan pasang kota yang berbeda (All-pairs Shortest Path Problems)

Untuk memecahkan masing-masing dari masalah-masalah tersebut terdapat sejumlah solusi.
- Algoritma Dijkstra untuk Single-source Shortest Path
- Algoritma Floyd-Warshall untuk masalah All-pairs Shortest Path

• Algoritma Johnson untuk masalah All-pairs Shortest Path pada sparse graph
Dalam beberapa masalah graph lain, suatu graph dapat memiliki bobot negatif dan kasus ini dipecahkan oleh algoritma Bellman-Ford. Yang akan dibahas di sini adalah algoritma Dijkstra yaitu mencari lintasan terpendek dari suatu verteks asal tertentu vs ke setiap vertek.

DAFTAR PUSTAKA

Andri Kristanto, *Algoritma & Pemrograman dengan C++ Edisi 2*, Graha Ilmu, Yogyakarta, 2009.

Budi Raharjo, *Pemrograman C++*, Informatika, Bandung, 2010.

Moh. Sjukani, *Algoritma & Struktur Data dengan C, C++ dan JAVA*, Mitra Wacana Media, Bandung, 2004.

Moh. Sjukani. Algoritma (Algoritma dan Struktur Data 1) dengan C, C++, dan Java. Mitra Wacana Media, 2007