BAB II

TEORI PENUNJANG

2.1. Dasar - Dasar Algoritma

Terbentuknya model matematika yang sesuai pada suatu pemecahan masalah, hanya merupakan bagian dari proses pemecahan masalah tersebut. Untuk melengkapi dalam memecahkan suatu masalah diperlukan suatu metode yang akan memecahkan masalah-masalah umum yang menggunakan model matematik. Idealnya, dalam memecahkan suatu masalah diperlukan suatu prosedur yang berisi urutan langkah-langkah yang berperanan penting pada suatu jawaban yang diinginkan.

Sebuah Algoritma adalah prosedur tertentu untuk menyelesaikan sebuah permasalahan dengan langkah dalam jumlah terbatas. Istilah algoritma pertama kali ditulis oleh *al-Khowarizmi* matematikawan arab abad 19, dalam bukunya *Kitab al-jabr w'al muquabala.*

Ada beberapa sifat umum algoritma. Sifat - sifat tersebut digunakan untuk mempermudah dalam mengingat ketika sebuah algoritma dideskripsikan. Sifat - sifat tersebut adalah :

1. *Input* (masukan). Suatu algoritma dapat mempunyai nol atau banyak input, yaitu suatu kuantitas yang diberikan nilai awal sebelum algoritma dimulai. Input diperoleh dari himpunan objek yang telah ditetapkan.
2. *Output* (keluaran). Dari setiap nilai input sebuah algoritma menghasilkan nilai output / keluaran dari himpunan yang dispesifikasikan. Nilai Output adalah penyelesaian dari permasalahan. Suatu algoritma dapat mempunyai satu atau banyak output.

3. *Definiteness* (tertentu). Langkah - langkah sebuah algoritma harus didefinisikan dengan tepat; tindakan penyelesaian harus ditetapkan dengan tepat dan jelas untuk setiap kasus.
4. *Finiteness* (terbatas). Dengan langkah yang terbatas (berakhir) sebuah algoritma harus dapat menghasilkan output yang diinginkan setelah mencapai akhir langkah-langkah untuk sebarang himpunan input.
5. *Effectiveness* (efektif). Suatu algoritma harus memungkinkan untuk setiap langkah-langkahnya dilakukan dengan jelas/pasti dan dalam jumlah waktu yang terbatas.

2.2. Pengurutan (Sorting)

Sorting (pengurutan) biasanya diketahui sebagai suatu proses penyusunan/pengurutan yang diberikan pada suatu obyek dengan menggunakan perintah-perintah tertentu. Tujuan penyortiran adalah untuk mempermudah dalam proses pencarian dari data satu ke data yang lain dari suatu obyek. Proses penyortiran sangat erat hubungannya dengan pengolahan data, bahkan merupakan bagian yang sangat penting dari pengolahan data.

Secara umum sorting dibagi menjadi dua katagori yaitu :

1. Sorting of array (larik)
2. Sorting of (sequential) files (file / arsip)

Keduannya sering disebut sebagai internal sorting dan eksternal sorting. Disebut internal sorting karena datanya berada pada array yang tersimpan dalam internal memori komputer atau biasa disebut RAM (Random Access Memory), sedangkan

eksternal sorting datanya tersimpan dalam fail-fail yang tersimpan dalam eksternal memori komputer misalnya disket, hard disk, compact disk dan lain-lain.

Internal sorting dibagi menjadi tiga yaitu :

1. Sorting by insertion (penyisipan)
2. Sorting by selection (penyeleksian)
3. Sorting by exchange (penukaran)

dan yang termasuk dalam eksternal sorting misalnya : mergesort dan polyphasesort. Dalam hal ini Quicksort termasuk dalam sorting by exchange.

Quicksort sebagai bagian dari sorting by exchange merupakan algoritma pengurutan yang paling efektif dari algoritma-algoritma pengurutan lain. Keefektifan ini dapat dilihat dari rata-rata Running Time (waktu yang diperlukan oleh suatu program untuk menghasilkan output) yang lebih kecil dari algoritma pengurutan yang lain, yaitu O(n log n). Rata-rata running time tersebut akan lebih efektif jika data yang diurutkan dalam jumlah besar dan jika data yang diurutkan dalam jumlah kecil, Running Time $O(n^2)$ akan lebih efektif dari running time O(n log n).

Algoritma Quicksort ini tidak hanya sederhana, tapi secara empiris dan analitis menunjukkan bahwa Quicksort dapat dua kali lebih cepat dari algoritma yang lain. Karena sebab-sebab di atas maka penggunaan Quicksort dapat dikembangkan dalam implementasi-implementasinya. Dalam implementasinya metode Quicksort dapat dikembangkan dengan melakukan perbaikan-perbaikan serta penambahan suatu algoritma sehingga menjadi algoritma sorting yang lebih efektif dan efisien.

2.3. Bahasa Pemrograman Pascal

Pascal adalah bahasa tingkat tinggi (high level language) yang orientasinya pada segala tujuan, dirancang oleh Prof. Niklaus Wirth dari Technical University di Zurich, Switzerland. Nama Pascal diambil sebagai penghargaan terhadap Blaise Pascal, ahli Matematik dan Philosophi terkenal abad 17 dari Prancis.

Prof. Niklaus Wirth memperkenalkan compiler bahasa Pascal pertama kali untuk komputer CDC 6000 (Control Data Corporation) yang dipublikasikan pada tahun 1971 dengan tujuan untuk membantu mengajar program komputer secara sistematis khususnya untuk memperkenalkan pemrograman yang terstruktur (structured programming). Jadi Pascal adalah bahasa yang ditujukan untuk membuat program terstruktur.

Dalam waktu singkat Pascal telah menjadi bahasa yang populer dikalangan pelajar Universitas dan merupakan bahasa yang diajarkan di beberapa Perguruan Tinggi.

2.3.1. Statemen Perulangan

Perulangan (loop) merupakan bentuk yang sering ditemui dalam suatu program aplikasi. Bahasa Pascal mengenal tiga macam statemen perulangan yaitu statemen **for, While-Do** dan **Repeat .... Until**.

2.3.1.1. Statemen Perulangan For

Perulangan dengan statemen **For** digunakan untuk mengulang statemen atau satu blok statemen sejumlah tertentu. Bentuk-bentuk perulangan **For** terdiri dari perulangan positif, perulangan negatif, dan perulangan tersarang.

Perulangan Positip

Perulangan positip adalah perulangan dengan penghitung (counter) dari kecil ke besar atau pertambahannya positip. Bentuk umum perulangan positip adalah :

FOR variabel-kontrol := nilai-awal **TO** nilai-akhir **DO** statemen

Variabel-kontrol, nilai-awal dan nilai akhir harus mempunyai tipe yang sama yaitu Integer. Untuk bagian statemen dapat berupa satu statemen ataupun dalam bentuk blok statemen (diawali dengan **Begin** dan diakhiri dengan **End**).

Perulangan Negatip

Perulangan negatip adalah perulangan dengan penghitung (counter) dari besar ke kecil atau pertambahannya negatip. Bentuk umum perulangan negatip adalah :

FOR variabel-kontrol := nilai-awal **DOWNTO** nilai-akhir **DO** statemen

Variabel-kontrol, nilai-awal dan nilai akhir harus mempunyai tipe yang sama yaitu Integer. Untuk bagian statemen dapat berupa satu statemen ataupun dalam bentuk blok statemen (diawali dengan **Begin** dan diakhiri dengan **End**).

Perulangan Tersarang

Perulangan tersarang (nested loop) adalah perulangan yang berada pada perulangan yang lainnya, dengan perulangan yang lebih dalam akan diproses terlebih dahulu sampai habis, kemudian perulangan yang lebih luar baru akan bertambah, kemudian mengerjakan perulangan yang lebih dalam lagi mulai dari nilai awal, dan seterusnya. Perulangan tersarang banyak digunakan dalam aplikasi matrik dengan menggunakan variabel tipe larik.

Contoh penggunaan statemen perulangan For adalah :

```
for I := 1 to 4 do
    begin
        write(I);
        writeln(' Pascal');
    end;
```

hasil eksekusi program :

1 Pascal

2 Pascal

3 Pascal

4 Pascal

2.3.1.2. Statemen Perulangan While-Do

Statemen **While-Do** digunakan untuk melakukan proses perulangan suatu statemen atau blok statemen secara terus menerus selama kondisi ungkapan-logika pada **While** masih bernilai logika benar. Statemen **While-Do** cocok untuk perulangan yang

jumlahnya tidak tentu (belum diketahui). Bentuk umum perulangan **While-Do** adalah sebagai berikut :

```
While kondisi Do
Begin
    Statemen_1;
    Statemen_2;
    .....
    .....
end;
```

Contoh :

```
I := 1;
While I <= 3 Do
Begin
    writeln(I);
    I := I + 1;
end;
```

hasil eksekusi program :

1

2

3

Perulangan While-Do Tersarang

Perulangan **While-Do** tersarang (nested **While-Do**) merupakan perulangan **While-Do** yang berada di dalam perulangan **While-Do** yang lainnya. Proses perulangan **While-Do** tersarang dimulai dengan test kondisi ungkapan logika pada **While-Do** terluar jika terpenuhi dilanjutkan dengan test kondisi ungkapan logika pada **While-Do** dalam jika terpenuhi maka perulangan **While-Do** dalam akan diproses sampai selesai, baru kemudian melanjutkan proses pada perulangan **While-Do** luar dan seterusnya sampai kondisi ungkapan logika pada **While-Do** luar tidak terpenuhi.

Bentuk umum perulangan **While-Do** tersarang adalah sebagai berikut :

```
While kondisi_1  Do
Begin
        While kondisi_2  Do
        begin
                Statemen_21;
                Statemen_22;
                .....
        end;
Statemen_11;
Statemen_12;
.....
end;
```

Contoh :

```
I := 1;
```

```
J:= 1;
While I <= 3 Do
Begin
        While J >= 3 - I Do
        Begin
                writeln('PASCAL');
                J := J - 1;
        end;
write(I);
I := I + 1;
end;
```

hasil eksekusi program :

1 PASCAL

2 PASCAL

3

2.3.1.3. Statemen Perulangan Repeat ... Until

Perulangan **Repeat** ... **Until** digunakan untuk mengulang (**Repeat**) suatu statemen atau blok statemen sampai (**until**) kondisi yang diseleksi setelah **Until** terpenuhi. Bentuk umum perulangan **Repeat ... Until** adalah sebagai berikut :

```
Repeat
    statemen_1;
```

```
    statemen_2;
    .....
Until kondisi;
```

Contoh :

```
I := 0;
Repeat
    I := I + 1;
    writeln(I);
until I >= 3;
```

hasil eksekusi program :

1

2

3

Perbedaan antara perulangan **Repeat ... Until** dengan perulangan **While-Do** adalah :

- Paling sedikit statemen-statemen di dalam perulangan **Repeat ... Until** akan diproses sekali sedangkan dalam perulangan **While-Do** nol kali.
- Pada perulangan **Repeat ... Until** proses perulangan pada blok statemen tidak memerlukan statemen **Begin** dan **End** sebagai batas blok statemen, sebaliknya dalam perulangan **While-Do** harus menggunakan statemen **Begin** dan **End**.

Perulangan Repeat ... Until Tersarang

Perulangan **Repeat ... Until** tersarang (nested **Repeat ... Until**) merupakan perulangan **Repeat ... Until** yang berada di dalam perulangan **Repeat ... Until** yang lainnya. Proses perulangan **Repeat ... Until** tersarang dimulai dengan masuk ke dalam perulangan **Repeat ... Until** luar yang dilanjutkan ke **Repeat ... Until** dalam. Proses selanjutnya adalah menyelesaikan proses **Repeat ... Until** dalam yaitu sampai kondisi yang diseleksi di **Until** dalam terpenuhi. Proses **Repeat ... Until** luar akan dilanjutkan setelah keluar dari **Repeat ... Until** dalam dan seterusnya sampai kondisi yang diseleksi di **Until** luar terpenuhi.

Bentuk umum perulangan **Repeat ... Until** tersarang adalah sebagai berikut :

```
Repeat
    Repeat
        statemen_21;
        statemen_22;
    .....
    Until kondisi_2;
    statemen_11;
    statemen_12;
    .....
Until kondisi_1;
```

Contoh :

```
I := 0;
J := 0;
```

```
Repeat
    Repeat
        J := J + 1;
        Write('PASCAL ');
    Until J >= I;
    I := I + 1;
    writeln(I);
until I >= 3;
```

hasil eksekusi program :

PASCAL 1

PASCAL 2

PASCAL 3

2.3.2. Statemen Penyeleksian Kondisi

Statemen penyeleksian kondisi menunjukkan bahwa suatu statemen akan dikerjakan bila suatu kondisi adalah benar. Jika kondisinya salah, maka statemen yang lainnya atau statemen setelah kata cadangan **ELSE** yang akan dikerjakan. Khusus untuk statemen **ELSE**, penulisan statemen sebelum **ELSE** tidak boleh diakhiri dengan titik koma, karena titik koma menunjukkan akhir statemen, padahal statemen tersebut masih satu rangkaian. Bentuk umum statemen penyeleksian kondisi adalah :

```
If kondisi then
    statemen_1
else
```

statemen_2;

Statemen yang digunakan untuk penyeleksian kondisi selain statemen **IF-THEN** juga statemen **CASE-OF**. Bentuk umum dari statemen **CASE-OF** adalah :

Case variabel of

konstanta_1 : statemen_1;

konstanta_2 : statemen_2;

.....

end;

Tipe data dari konstanta harus sama dengan tipe data dari variabel serta tidak boleh bertipe real. Statemen yang mempunyai nilai konstanta yang sama dengan variabel akan diproses, sedang statemen yang lainnya tidak.

2.3.3. Prosedur

Prosedur adalah suatu program terpisah dalam blok tersendiri yang berfungsi sebagai sub program (program bagian). **Prosedur** diawali dengan kata cadangan **PROCEDURE** di dalam bagian deklarasi prosedur. **Prosedur** dipanggil dan digunakan dalam blok program lainnya dengan menyebutkan judul prosedurnya.

Tujuan penggunaan **prosedur**, terutama pada program terstruktur adalah :

1. Untuk memecah suatu program yang besar dan rumit menjadi bagian-bagian yang lebih kecil dan sederhana.
2. Untuk menuliskan sekumpulan instruksi yang sering dilakukan berkali-kali hanya dengan sekali tulis saja.

Hal-hal yang perlu diketahui untuk dapat membuat suatu **prosedur** adalah :

a. Nilai-nilai parameter atau variabel di dalam suatu modul atau **prosedur** program Pascal bersifat lokal, artinya parameter atau variabel tersebut hanya berlaku di dalam **prosedur** tersebut, bila parameter atau variabel tersebut di gunakan atau dioprasikan di luar **prosedur** maka program akan menunjukkan kesalahan (variabel belum didefinisikan) pada saat program di jalankan (walaupun parameter atau variabel tersebut sudah didefinisikan pada suatu **prosedur**).

b. Supaya nilai variabel dalam suatu **prosedur** bersifat global, maka variabel tersebut harus dideklarasikan di atas **prosedur** yang akan menggunakannya.

c. Nilai-nilai variabel dalam **prosedur** dapat dikirimkan dari dan ke modul utama, pengiriman parameter dari dan ke modul utama dapat dilakukan dengan cara :

c.1. Pengiriman parameter secara nilai

Parameter-parameter yang telah dikirim dari modul utama ke suatu pro **sedur** akan menjadi parameter lokal dalam **prosedur** tersebut, sehingga prubahan nilai-nilai parameter dalam **prosedur** tidak akan mengubah nilai parameter tersebut dalam modul utama. jadi sifatnya satu arah.

Bentuk umum :

```
Procedure pengenal(par_1,par_2,..... : tipe_data);
```

c.2. Pengiriman parameter secara acuan.

Perubahan-perubahan dalam **prosedur** akan dapat mengubah nilai-nilai parameter modul utama. Jadi sifatnya dua arah, yaitu masuk dan keluar.

Bentuk umum :

```
Procedure pengenal(var par_1,par_2,..... : tipe_data);
```

par_1, par_2, ... adalah parameter yang diterima / dikirimkan.

Pengiriman parameter secara nilai dan acuan dapat digunakan secara bersama-sama.

d. **Prosedur** yang telah dibuat harus dipanggil jika akan digunakan pada modul program lainnya. Bentuk umum pemanggil **prosedur** adalah :

Pengenal(parameter_1,parameter_2, ...);

Pengenal adalah nama **prosedur** yang dideklarasikan sebelumnya. Parameter_1, Parameter_2, ... adalah daftar parameter yang harus memenuhi syarat berikut :

- Urutan, jumlah dan jenis datanya harus sama dengan urutan, jumlah dan jenis data pada **prosedur** yang dipanggilnya.
- Nama-nama dari masing-masing parameter dapat berbeda dengan nama-nama parameter pada **prosedur** yang dipanggilnya.
- Argumen pemisah parameter berupa koma.
- Untuk pemanggilan **prosedur** secara acuan kata cadangan **VAR** tidak lagi dipakai.

2.3.4. Fungsi

Blok **fungsi** hampir sama dengan blok **prosedur**, hanya **fungsi** harus dideklarasikan dengan tipenya. Tipe deklarasi ini menunjukkan tipe hasil dari **fungsi**. Tipe tersebut ditulis pada akhir deklarasi **fungsi** yang didahului dengan tanda titik dua (:), sebagai berikut :

Function pengenal(par_1,par_2,..... : tipe_data): tipe_hasil;

contoh :

Function Faktorial(Var fak, hasil : integer): integer;

Function Pangkat(X,Y : real): real;

Blok **fungsi** sama dengan blok **prosedur** yang diawali dengan **Begin** dan diakhiri dengan **End.**

Perbedaan **fungsi** dengan **prosedur** adalah :

1. Pada **fungsi**, nilai yang dikirimkan balik terdapat pada nama **fungsinya** (kalau pada **prosedur** pada parameter yang dikirimkan secara acuan).
2. Karena nilai balik di nama **fungsi** tersebut, maka **fungsi** tersebut dapat langsung digunakan untuk dicetak hasilnya atau nilai **fungsi** tersebut dapat juga langsung dipindahkan ke pengenal variabel yang lainnya, sedang pada **prosedur**, nama **prosedur** tersebut tidak dapat digunakan langsung, yang dapat digunakan langsung adalah parameternya yang mengandung nilai balik.

2.3.5. Array

Array adalah sebuah struktur homogen, terdiri dari komponen-komponen yang seluruhya bertipe data sama. Nilai salah satu komponen dapat diambil dengan menyebutkan indeks komponen tersebut. **Array** disebut juga sebagai struktur random akses, karena semua komponen dapat dipilih secara acak dan dapat diakses secara sama.

Perhatikan deklarasi variabel dalam Pascal berikut :

Pengenal : array [tipe_indeks] of tipe_dasar_pengenal;

Pernyataan ini mendeklarasikan pengenal sebagai rangkaian sel yang berurut satu dimensi. Untuk menotasikan sebuah komponen individu, nama dari pengenal diperluas dengan menyebutkan indeks pemilihan komponennya. Indeks ini menjadi sebuah harga dari tipe yang didefinisikan sebagai tipe_indeks.

Contoh penggunaan **array** :

nilai = (56 42 89 65 48)

jadi :

nilai[1] menuju ke 56

nilai[2] menuju ke 42

nilai[3] menuju ke 89

nilai[4] menuju ke 65

nilai[5] menuju ke 48

2.3.6. Rekursi

Rekursi (recursion) adalah proses dari suatu sub program (dapat berupa fungsi maupun prosedur) yang memanggil dirinya sendiri. Proses **rekursi** untuk beberapa kasus merupakan algoritma (algorithm) yang baik dan dapat membuat pemecahan masalah lebih mudah. Akan tetapi sebagai imbalannya, proses **rekursi** ini harus dibayar mahal dengan memori yang banyak digunakan, dikarenakan setiap kali suatu sub program dipanggil, maka diperlukan sejumlah tambahan memori.

contoh :

```
Function faktorial ( n : byte) : longint;
```

```
begin
    if n <= 1 then faktorial := 1
    else faktorial := n*faktorial (n-1);
end;
```

2.3.7. Grafik

Unit standar **Graph** menyediakan suatu pustaka lebih dari 50 buah rutin grafik yang dapat dipergunakan untuk keperluan pembuatan grafik. Untuk membuat grafik dengan fasilitas ini maka unit standar **Graph** harus disebutkan dalam program. Disamping file *TURBO.TPL* yang berisi unit standar **Graph**, satu atau lebih file dengan ekstension *.BGI* yang merupakan *grafik driver* juga harus ada. *Grafik driver* ini menunjukkan *Graphics Adapter* yang digunakan untuk monitor.

Menggunakan Grafik

Untuk memulai menggunakan grafik, maka prosedur standar *Initgraph* harus disebutkan terlebih dahulu atau dengan menyebut prosedur tertentu, yaitu prosedur yang memiliki sintak sama dengan prosedur standar Inithgrap tersebut dibawah :

```
Initgraph ( Var  Graph_Driver : integer;
            Var  Graph_Mode  : integer;
                 Driver_Path   : string );
```

Graph_Driver merupakan driver yang dipergunakan pada komputer. Driver tersebut dapat dipilih sesuai dengan *graphics adapter* yang digunakan komputer. Jika tidak mengetahui *graphics adapter* yang dipakai, dapat dilakukan pendeteksian secara

otomatis (*autodetection*) oleh Turbo Pascal yaitu dengan menggunakan konstanta *Detect* atau nilai 0 (nol).

Graph_Mode adalah mode grafik yang digunakan untuk driver yang bersangkutan. Jika dalam *Graph_Driver* menggunakan *autodetection* maka mode grafik juga akan dipilih secara otomatis, sebaliknya jika tidak menggunakan *autodetection* maka *Graph_Mode* juga harus disebutkan.

Driver_Path adalah menunjukkan direktori letak dari file *graphics driver* (file yang berekstension *.BGI*) di disk.

Untuk berpindah dari keadaan mode grafik ke mode teks dapat dilakukan dengan menggunakan prosedur standar *RestoreCrtMode* dan untuk kembali ke keadaan mode grafik menggunakan prosedur standar *SetGraphMode*.

Untuk mengakhiri penggunaan grafik dan kembali pada keadaan mode layar semula (sebelum mode grafik dipergunakan) digunakan prosedur standar *CloseGraph*.