

BAB II

LANDASAN TEORI

2.1. Antrian (*Queue*)

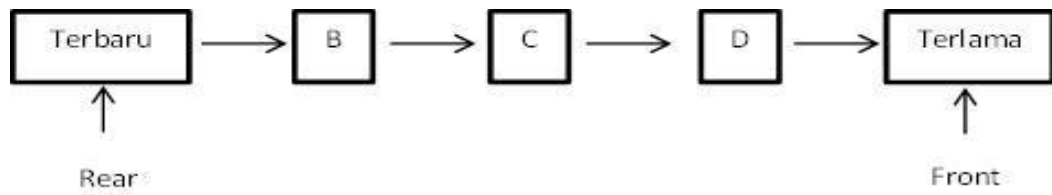
Antrian sering dijumpai dalam kehidupan sehari-hari contohnya rumah sakit, pembelian tiket nonton bioskop, pembelian tiket kereta dan lain lain.

Antrian atau *Queue* (baca: *qyu*) adalah salah satu struktur data yang memiliki sistem kerja pertama masuk dan pertama keluar (FIFO = *First In First Out*) seperti halnya antrian pada dunia nyata (Salahuddin, 2010). Antrian atau *Queue* atau lebih dikenal dengan struktur FIFO (*First In First Out*) merupakan salah satu tipe data abstrak yang sering digunakan dalam ilmu komputer (Wibowo, 1989). *Queue* pada penambahan datanya hanya dapat dilakukan pada salah satu ujung dan penghapusan data hanya dapat dilakukan pada ujung yang lain. Elemen baru dumasukan pada bagian belakang dalam antrian dan penghapusan elemen hanya dilakukan pada bagian ujung depan elemen.

Ada berbagai jenis model antrian, antara lain :

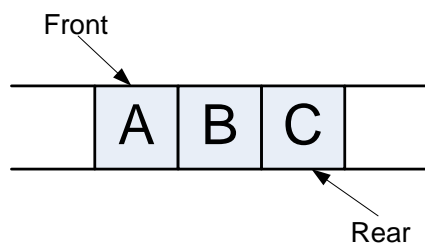
1. *First In First Out (FIFO)* : Dimana orang yang datang lebih awal akan diberikan pelayanan terlebih dahulu, orang yang datang terakhir akan dilayani terakhir. Contoh : Pelayanan kasir supermarket.
2. *Last In First Out (LIFO)* : Model antrian dimana yang datang terakhir dilayani lebih awal, dan yang datang paling awal dilayani terakhir. Contoh : Tumpukan barang di gudang, dimana tumpukan paling atas merupakan barang terakhir yang masuk, sehingga akan diambil pertama.
3. *Priority Service (PS)* : Model antrian yang mengutamakan layanan kepada antrian yang memiliki prioritas lebih tinggi dibandingkan dengan prioritas yang lebih rendah. Meskipun datang paling akhir, jika prioritasnya paling tinggi maka akan didahulukan.

Antrian mengenal dua prosedur utama yaitu memasukan elemen baru dan mengeluarkan/menghapus elemen yang sudah ada (Sumantri, 1988). Dimana elemen baru dimasukan sebagai elemen terakhir dalam antrian yang dikenal dengan prosedur *add/insert*. Elemen yang dihapus/dikeluarkan adalah elemen pertama yang dikenal dengan prosedur *delete/remove*.



Gambar 2.1 Antrian/Queue

Berikut ini adalah operasi pada *queue* :



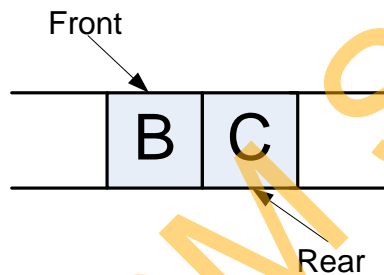
Proses penambahan data pada

Queue

`q.insert(A);`

`q.insert(B);`

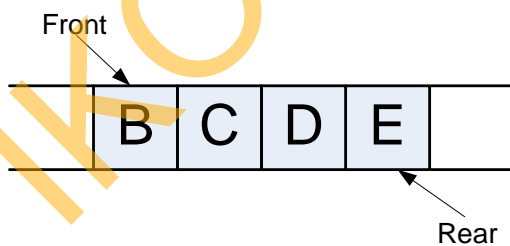
`q.insert(C);`



Proses penghapusan data pada

Queue

`x= q.remove();`



Proses penambahan data pada

Queue, data yang dimasukan

akan ada pada posisi terakhir

`q.insert(D);`

`q.insert(E);`

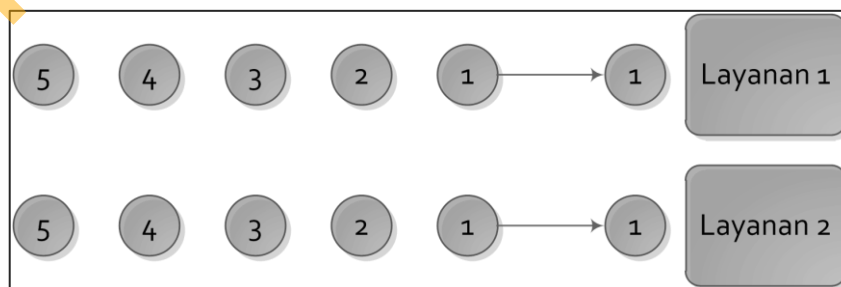
1. Antrian datang secara berurutan A, B dan C yang kemudian dimasukan ke dalam baris antrian. Urutan data masuk melalui bagian *rear* antrian.

2. Setelah A dilayani, maka A akan keluar dari bagian *Front* baris antrian.
3. Kemudian ada antrian baru yang masuk yaitu D dan E secara berurutan masuk melalui bagian *rear* lagi. Dan seterusnya.

Dalam proses bisnis, terdapat berbagai contoh dari berbagai proses yang menciptakan/menimbulkan antrian. Menurut Suad Husnan (1982), terdapat model antrian yang berbentuk antrian secara *single channel*/satu jalur dan antrian secara *multiple channel*/lebih dari satu jalur :

1. *Single Channel Queue*

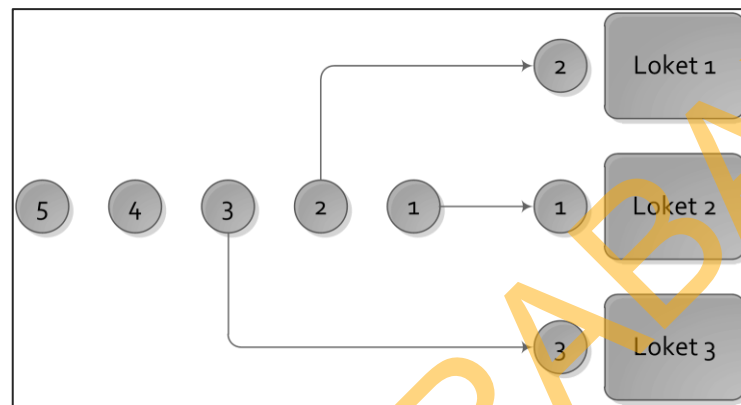
Merupakan bentuk antrian yang paling sederhana. *Single channel* menyatakan bahwa hanya ada satu fasilitas pelayanan atau dapat juga dikatakan untuk memasuki sistem tersebut hanya ada satu jalur. Atau lebih dari satu jalur tunggu dengan pelayanan yang berbeda-beda.



Gambar 2.2 *Single Channel Queue*

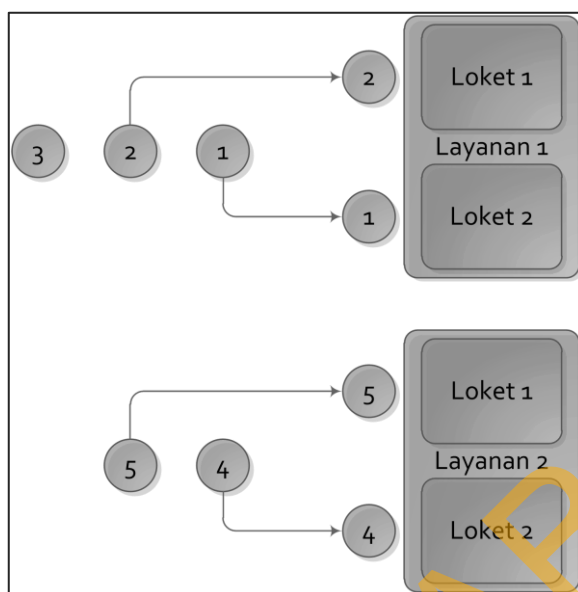
2. Multiple Channel Queue

Multiple channel queue merupakan suatu model antrian yang memiliki jumlah fasilitas pelayanan lebih dari satu *channel* pelayanan.



Gambar 2.3 *Multiple Channel Queue* satu jenis layanan

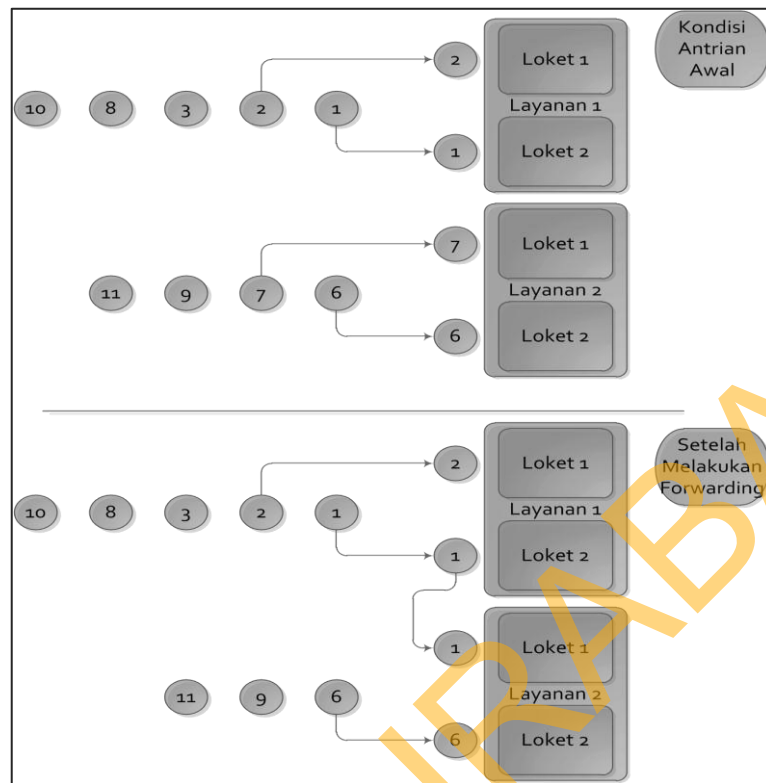
Adapun *multiple channel queue* dengan lebih dari satu jenis layanan, dimana seseorang mengambil nomor antrian sesuai dengan urutan antrian namun pada jalur tunggu pada jenis layanan yang berbeda.



Gambar 2.4 *Multiple Channel Queue* dengan Pelayanan Yang Berbeda

3. *Multiple Channel Queue* Dengan *Forwarding*

Merupakan antrian merupakan penerapan dari *multiple channel queue* dengan lebih dari satu jenis layanan dengan kemampuan memforward sebuah nomor ke jenis layanan yang berbeda tanpa perlu mengambil nomor antrian lagi dan diberikan prioritas sesuai dengan nomor antrianya. Dimana nomor antrian yang lebih kecil akan tetap mendapat pelayanan lebih dahulu. Ini merupakan implementasi dari *multiple channel queue* dimana terdapat multi *server* atau multi layanan yang mana jenis layanan lebih dari 1 layanan.



Gambar 2.5 *Multiple Channel Queue Dengan Forwarding*

2.2. Konsep Dasar Sistem


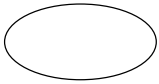
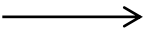
Sistem adalah kumpulan elemen yang saling terkait dan bertanggung jawab memproses masukan (*input*) sehingga menghasilkan keluaran (*output*) (Soehandoro, 2004). Elemen-elemen sistem antara lain :

1. Tujuan, adalah hal yang ingin dicapai dari sistem tersebut berupa tujuan usaha, kebutuhan, masalah, prosedur pencapaian tujuan.

2. Batasan, adalah batasan-batasan yang ada dalam mencapai tujuan dari sistem. Dapat berupa peraturan-peraturan, biaya-biaya, peralatan dan lain-lain.
3. Kontrol, adalah pengawas dari pelaksanaan pencapaian tujuan sistem yang dapat berupa kontrol masukan data, keluaran dan pengoperasian.
4. *Input*, adalah bagian dari sistem yang menerima data masukan.
5. Proses, adalah bagian dari sistem yang memproses data menjadi informasi sesuai dengan keinginan penerima berupa klarifikasi, peringkasan dan pencarian.
6. *Output*, adalah bagian dari sistem yang bertugas menampilkan keluaran atau tujuan akhir dari sistem.
7. Umpan balik, berupa perbaikan dan pemeliharaan.

2.3. Use Case Diagram

Diagram *use case* atau *use case* diagram menyajikan interaksi antara *use case* dan aktor (Sholiq, 2006). Dimana, aktor merupakan orang, peralatan atau sistem lain yang berinteraksi dengan sistem yang sedang dibangun. *Use case* menggambarkan fungsionalitas sistem atau persyaratan-persyaratan yang harus dipenuhi sistem dari pandangan pemakai. Komponen dari *use case* diagram antara lain :

1	<i>Bussiness Actor</i>		Mempresentasikan seseorang atau sesuatu (seperti perangkat, sistem lain) yang berinteraksi dengan sistem.
2	<i>Use Case</i>		Gambaran fungsionalitas dari suatu sistem, sehingga <i>customer</i> atau pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun.
3	<i>Flow Event</i>		Menunjukkan aliran <i>event</i> .

Elemen-elemen yang ada di dalam *use case* diagram dihubungkan dengan relasi-relasi yang menunjukkan interaksi antar elemen. Ada beberapa relasi yang terdapat pada *use case* diagram:

1. *Association*, menghubungkan link antar element.
2. *Generalization*, disebut juga *inheritance* (pewarisan), sebuah elemen dapat merupakan spesialisasi dari elemen lainnya.
3. *Dependency*, sebuah element bergantung dalam beberapa cara ke elemen lainnya.
4. *Aggregation*, bentuk *association* dimana sebuah elemen berisi elemen lainnya.

Tipe relasi/ *stereotype* yang mungkin terjadi pada *use case diagram*:

1. <<*include*>>, yaitu kelakuan yang harus terpenuhi agar sebuah *event* dapat terjadi, dimana pada kondisi ini sebuah *use case* adalah bagian dari *use case* lainnya.
2. <<*extends*>>, kelakuan yang hanya berjalan di bawah kondisi tertentu seperti menggerakkan alarm.
3. <<*communicates*>>, mungkin ditambahkan untuk asosiasi yang menunjukkan asosiasinya adalah *communicates association*. Ini merupakan pilihan selama asosiasi hanya tipe *relationship* yang dibolehkan antara *actor* dan *use case*.

2.4. Entitas Relationship Diagram (ERD)

Entity Relationship Diagram (ERD) dan dikenal juga dengan *Entity Relationship Model* (ERM) adalah sebuah model konseptual dari data yang menggambarkan keadaan sebenarnya dari *entities* dan *relationships* sebagai suatu cara untuk menggambarkan *relational database* (Wahyudi, 2008). Dengan menggunakan ERD ini, dapat dilihat dengan jelas hubungan antar *file-file database* dan melalui ERD ini seorang *programmer* diharapkan dapat menentukan seperti apakah program yang akan dibuat nantinya.

Entity atau Entitas adalah sebuah objek yang ada di suatu unit usaha yang akan dibuat komputerisasinya, atau entitas adalah suatu objek yang unik yang bisa dibedakan antara satu objek dengan objek lainnya (Wahyudi, 2008). *Attribute* atau Atribut adalah karakteristik yang biasa untuk menggambarkan seluruh atau sebagian dari *record* (Wahyudi, 2008). Nilai Atribut merupakan suatu data aktual atau informasi yang disimpan pada suatu atribut di dalam suatu *entity* atau *relationship*. Jenis-jenis atribut :

1. *Key*, Atribut yang digunakan untuk menentukan suatu *entity* secara unik. Contoh : nomor induk pegawai.
2. Atribut *Simple* : Atribut yang bernilai tunggal. Contoh : nama, tanggal lahir
3. Atribut *Multivalued* : Atribut yang memiliki sekelompok nilai untuk setiap *instant entity*. Contoh : gelar, nomor telepon.
4. Atribut *Composite* : Suatu atribut yang terdiri dari beberapa atribut yang lebih kecil yang mempunyai arti tertentu. Contoh : Nama yang dibagi menjadi nama depan, nama tengah dan nama belakang.
5. Atribut *Derivatif* : Suatu atribut yang dihasilkan dari atribut yang lain. Contoh : tanggal lahir menghasilkan atribut umur.

Relationship adalah keterhubungan atau keterkaitan antara entitas satu dengan entitas lainnya. *Cardinality ratio constraint* merupakan menjelaskan batasan jumlah keterhubungan satu *entity* dengan *entity* lainnya. Terdapat tiga Jenis *cardinality ratio constraints*, satu pada satu (1:1), satu pada banyak (1:N/ N:1) dan banyak pada banyak (M:N).

1. One to one Relationship (1:1)

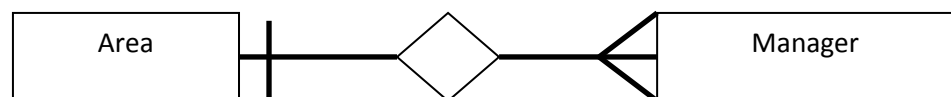
Hubungan antara *file* pertama dengan *file* kedua adalah satu berbanding satu.



Gambar 2.6 One-to-One Relationship

2. One to many relationship (1:N/ N:1)

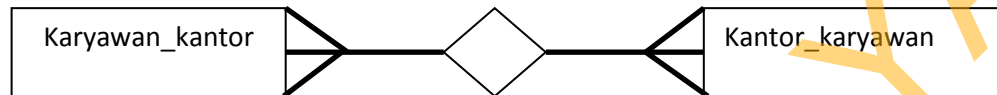
Hubungan antara *file* pertama dengan *file* kedua adalah satu berbanding banyak atau dapat pula dibalik, banyak lawan satu.



Gambar 2.7 One-to-Many Relationship

3. Many to many Relationship (M:N)

Hubungan antara *file* pertama dengan *file* kedua adalah banyak berbanding banyak.



Gambar 2.8 Many-to-Many Relationship

2.5.Database

Database adalah kumpulan *file-file* yang saling berelasi sehingga membentuk satu bangunan data untuk menginformasikan satu perusahaan atau instansi dalam batasan tertentu (Kristanto, 1993). Suatu Basis Data adalah koleksi data yang bisa mencari secara menyeluruh dan secara sistematis memelihara dan *me-retrieve* informasi (Simarmata, 2007). Komponen-komponen dari *database* antara lain :

1. *Entity*/Entitas, adalah orang, tempat, kejadian atau konsep yang informasinya direkam. Contoh untuk di Universitas : mahasiswa, mata kuliah, dosen, fakultas, jurusan dan lain-lain.
2. *Attribute*/Atribut, adalah sifat, perilaku atau ciri yang dimiliki oleh suatu entitas. *Attribute* juga disebut sebagai data elemen,

data field atau *data item*. Contoh atribut mahasiswa : nama, nim, jurusan, alamat, nama orang tua dan lain-lain.

3. *Data Value*, adalah data aktual atau informasi yang disimpan pada tiap data elemen atau *attribute*. *Data value* adalah isi dari *attribute*.
4. *Record/Tuple*, adalah kumpulan elemen-elemen yang saling berkaitan menginformasikan tentang suatu *entity* secara lengkap.

2.5.1. Relational Database Management System (RDBMS)

Relational Database Management System (RDBMS) atau Sistem Manajemen Basis Data Relasional adalah suatu istilah yang digunakan untuk menguraikan keseluruhan deretan program untuk mengelola sebuah basis data relasional dan komunikasi mesin basis data relasional (Simarmata, 2007).

Ada tiga prinsip dalam RDBMS :

1. *Data definition* : Mendefinisikan jenis data yang akan dibuat (dapat berupa angka atau huruf), cara relasi data, validasi data dan lainnya.

2. *Data Manipulation* : Data yang telah dibuat dan didefinisikan tersebut akan dilakukan beberapa pengerjaan, seperti menyaring data, melakukan proses *query*, dan sebagainya.
3. *Data Control* : Bagian ini berkenaan dengan cara mengendalikan data, seperti siapa saja yang bisa melihat isi data, bagaimana data bisa digunakan oleh banyak *user*, dan sebagainya.

Semua operasi *input* dan *output* yang berhubungan dengan *database* harus menggunakan DBMS. Bila pemakai akan mengakses *database*, DBMS menyediakan penghubung (*interface*) antara pemakai dengan *database*. Hubungan pemakai dengan *database* dapat dilakukan dengan dua cara :

1. Secara interaktif menggunakan bahasa pertanyaan (*query language*).
2. Dengan menggunakan program aplikasi.

2.5.2. SQL (Structured Query Language)

SQL singkatan dari *Structured Query Language*. SQL adalah bahasa yang digunakan untuk berkomunikasi dengan *database* (Irmansyah, 2003). SQL mula-mula didefinisikan oleh ISO (*International Standards Organization*) dan ANSI (*American National*

Standards Institute), bahasa ini merupakan standar untuk *Relational Database Management Systems* (RDBMS).

Pernyataan-pernyataan SQL digunakan untuk melakukan beberapa tugas seperti : *update* data pada *database*, atau menampilkan data dari *database*. Beberapa *software* RDBMS dan dapat menggunakan SQL, seperti : Oracle, Sybase, Microsoft SQL Server, Microsoft Access, Ingres, dan sebagainya. Setiap *software database* mempunyai bahasa perintah/*syntax* yang berbeda, namun pada prinsipnya mempunyai arti dan fungsi yang sama.

Perintah-perintah tersebut antara lain : "SELECT", "INSERT", "UPDATE", "DELETE", "CREATE", DAN "DROP", yang dapat digunakan untuk mengerjakan hampir semua kebutuhan untuk memanipulasi sebuah *database*. Di bawah ini adalah beberapa contoh perintah-perintah *query* dalam Microsoft SQL Server :

1. *Create* : digunakan untuk membuat tabel baru.

Contoh : *create table empinfo (Id varchar (5), first_name varchar(15), last_name varchar(20), address varchar(30), city varchar(20), state varchar(20));*

2. *Select* : digunakan untuk menampilkan data sesuai kriteria yang kita tentukan.

Contoh : *select first_name, last_name, city from empinfo where city <> 'Payson';*

3. *Insert* : digunakan untuk menyisipkan atau menambah baris pada tabel.

Contoh : *insert into empinfo (first_name, last_name, address, city, state) values ('Luke', 'Duke', '2130 Boars Nest', 'Peachtree', 'Georgia');*

4. *Update* : digunakan untuk mengupdate atau merubah isi data dalam tabel.

Contoh : *update empinfo set address = 'Jl.Bojong 12', city = 'Depok', State = 'West Java' where id=32382;*

5. *Delete* : digunakan untuk menghapus baris/record data dalam tabel

Contoh : *delete from empinfo where lastname = 'May';*

6. *Drop* : digunakan untuk menghapus tabel

Contoh : *drop table empinfo;*

2.6. Komunikasi Serial

Komunikasi serial adalah salah satu metode komunikasi data di mana proses pengiriman data dilakukan satu per satu sehingga diperlukan penghantar kirim data dan terima data (Ariyus, 2008).

Dimana, data dikirimkan dalam bentuk pulsa listrik yang disebut dengan *bit* secara berkelanjutan. Penerima juga menerima data dalam bentuk *bit* secara satu per satu secara berkelanjutan.

Komunikasi serial memiliki protokol komunikasi. Berikut ini adalah beberapa istilah dalam protokol komunikasi RS-232-C :

1. *Baud Periode*

Kecepatan transmisi data pada komunikasi serial dikenal dengan *Baud Periode* yang diukur dalam satuan *bit per second*.

Nilai-nilai *Baud Periode/Baud Rate* diantaranya : 50, 110, 300, 600, 1200, 2400, 4800, 9600, 19200.

2. *Marking state*.

Periode waktu selama tidak ada data yang dikirim. Selama kondisi *marking*, *output line* pengirim selalu *logic 1 (High)*.

3. *Start bit*.

Logika '0' (*Low*) menunjukkan transmisi data dimulai. Kondisi *low* yang terjadi pada *start bit* dinamakan *spacing state*.

4. Karakter *bit*.

Berisi data dengan jumlah 5, 6, 7 atau 8 *bit*. *Bit* pertama yang dikirim adalah LSB.

5. *Parity bit*

Parity bit adalah *bit* pilihan yang dikirim setelah karakter *bit* untuk mendeteksi *error* transmisi. Ada dua macam *parity* yaitu *parity* genap dan *parity* ganjil. Jika *parity* genap yang dipilih, *parity bit* yang berlogika T (*High*) sehingga jumlah *bit* logika T pada karakter *bit* dan *parity bit* adalah genap. Jika *parity* ganjil yang dipilih, *parity bit* membuat jumlah *bit* logika T pada karakter *bit* dan *parity bit* adalah ganjil. Jika terjadi *error* maka penerima akan *set error flag* pada *spesial register*.

6. *Stop bit*

Satu, satu setengah dan dua *bit* berlogika '1' (*High*) akan dikirim setelah karakter *bit* atau *parity bit* jika ada *parity bit*. Dengan adanya *stop bit* dapat dipastikan bahwa penerima mempunyai waktu yang cukup untuk menerima karakter berikutnya.

Hardware yang menggunakan prinsip komunikasi serial yang berhubungan dalam pengerjaan tugas akhir kali ini adalah *pole display* dan *display seven segment*. Namun kali ini penulis menggunakan *pole display*.



Gambar 2.9 *Pole Display*.

2.7. Realtime System

Dhamdhare (2003) mengungkapkan definisi tentang *Real Time Application* sebagai aplikasi yang memerlukan respon setiap saat dari komputer sistem untuk mencegah kegagalan komputasi. Ketika aplikasi melakukan *request*, komputer sistem harus melakukan komputasi dan memberikan hasil atau melakukan aksi atau menjalankan perintah yang diberikan sesuai *request* pada periode waktu tertentu (Dhamdhare, 2003). *Real Time System* digunakan jika suatu operasi memerlukan ketepatan waktu dari prosesor atau aliran data, dan sering digunakan sebagai pengontrol terhadap aplikasi-aplikasi tertentu (Kusumadewi, 2000). Dua bentuk *Real-Time System* yaitu :

1. *Hard Real-Time Task*, menjamin *critical task* dapat diselesaikan tepat pada waktunya, karena jika tidak akan mengakibatkan kerusakan dan kesalahan fatal pada sistem.
2. *Soft Real-Time task*, memberikan prioritas *critical task* dibandingkan dengan task yang lainya sehingga task tersebut dapat diselesaikan dengan segera. *Critical task* adalah proses yang dilakukan pada sebuah kejadian dan di operasikan secara berulang ulang dan terjadwal.

Pada bahasan ini, akan digukanan *Hard Real-Time* task karena perubahan data antrian yang tidak tentu membuat data nomor yang baru masuk, dipanggil dan sudah dilayani harus didata dengan cepat untuk menghindari kekacauan nomor antrian jika terjadi listrik mati.