1. ***Circular Linked List***

*Circular Linked List* adalah suatu *linked list* yang tidak memiliki nilai *nil/NULL* untuk medan sambungannya. Perhatikan Gambar 3.1 dan Gambar 3.2.

 Gambar 3.1 *Circular Single Linked List*

Gambar 3.2 *Circular Double Linked List*

Deklarasi bisa dilihat kembali di *Single Linked List* atau *Double Linked Li*st.

**III.1 *Circular Single Linked List***

Operasi-operasi yang ada pada *Circular Single Linked List* hampir sama seperti pada *Single Linked List* yang telah dibahas sebelumnya.

Operasi-operasinya sebagai berikut:

1. **Penciptaan**

Penciptaan adalah memberikan harga *nil* terhadap variabel pointer kepala (pointer Awal dan pointer Akhir), seperti terlihat pada Gambar 3.3.



Gambar 3.3 Hasil dari proses penciptaan *list*

1. **Penyisipan**

**2.a Penyisipan di depan/awal**

Operasi ini berguna untuk menambahkan satu simpul baru (data) di posisi pertama. Langkah pertama untuk penambahan simpul (data) di depan/awal dengan cara buat terlebih dahulu satu simpul baru, lalu mengisinya dengan data pada *field* info-nya. Pointer yang menunjuk ke simpul tersebut dipanggil dengan nama **baru**. Kondisi ketika akan menyisipkan simpul baru:

* + - ***Lis*t Kosong**
* Kondisi *list* ketika *list* masih kosong pointer Awal dan pointer Akhir masih berharga *nil* (Gambar 3.4).



Gambar 3.4 Kondisi list sebelum proses penyisipan

* Kondisi setelah operasi penyisipan

Operasi penyisipan pertama kali ketika *linked list* masih kosong adalah dengan mengisikan alamat pointer Baru ke pointer Awal dan pointer Akhir. Dan medan sambungan kanan (*next*) menunjuk ke simpul itu sendiri, seperti terlihat pada Gambar 3.5.

Akhir

Awal

Baru

|  |  |
| --- | --- |
| 1 |  |

Gambar 3.5 Keadaan *list* setelah proses penyisipan

* + - ***Lis*t tidak kosong**

Keadaan *list* mula-mula (Gambar 3.6):



Gambar 3.6 Keadaan *list* sebelum proses penyisipan di depan

Simpul yang akan yang berisi data 6 akan disisipkan di depan *circular linked list* yang sudah ada (Gambar 3.7):



Gambar 3.7 Simpul yang akan disisipkan

Maka keadaan *list* setelah penyisipan di depan seperti terlihat pada Gambar 3.8.



Gambar 3.8 Keadaan list setelah penyisipan di depan

**2.b. Penyisipan di tengah**

Penyisipan di tengah adalah proses menyisipkan simpul (data) di antara dua simpul (data).

Proses penyisipan di tengah sama seperti pada *single linked list* (lihat modul sebelumnya), karena tidak mengubah struktur *circular*-nya

* + - ***List* Kosong** (Sama dengan penyisipan di depan)
    - ***List* tidak kosong**

Mula-mula keadaan *circular linked list* (Gambar 3.9)



Gambar 3.9 Keadan *list* sebelum proses penyisipan

Akan menyisipkan simpul baru yang berisi data 4 setelah simpul yang berisi data 1 (Gambar 3.10)



Gambar 3.10 Simpul yang akan disisipkan

Keadaan *circular linked list list* setelah penyisipan di tengah seperti terlihat pada Gambar 3.11.



Gambar 3.11 Keadaan *list* setelah proses penyisipan di tengah

**2.c** **Penyisipan di akhir/belakang**

Penyisipan di akhir/belakang adalah proses menyisipkan simpul (data) setelah simpul (data) yang ditunjuk oleh pointer Akhir.

Keadaan *list* ketika akan disisipkan:

* + - ***List* kosong** (Sama dengan penyisipan di depan)
    - ***List* tidak kosong**

Mula-mula keadaan *circular linked list* (Gambar 3.12)



Gambar 3.12 Keadaan *list* sebelum adanya proses penyisipan

Simpul (data) yang akan disisipkan di akhir (Gambar 3.13):



Gambar 3.13 Simpul yang akan disisipkan

Keadaan *list* setelah simpul (data) disisipkan di akhir (Gambar 3.14):



Gambar 3.14 Keadaan *list* terjadi proses penyisipan di akhir

1. **Penghapusan**

Pada dasarnya proses penghapusan di *circular* sama seperti pada *linear linked list*.

Ada 3 kondisi yang perlu diperhatikan yaitu kondisi *linked list* masih kosong, kondisi *linked list* hanya memiliki satu data, dan kondisi *linked list* yang memiliki lebih dari satu data (satu elemen)

**Kondisi *linked list* kosong (**Pada kondisi ini proses penghapusan tidak bisa dilakukan, dan kondisi ini disimpan di luar dari prosedur ini)



**3.a. Penghapusan di awal/depan**

Penghapusan data di awal adalah proses menghapus simpul pertama (yang ditunjuk oleh variabel pointer Awal), sehingga variabel pointer Awal akan berpindah ke simpul berikutnya, dan medan sambungan kanan (*next*) dari simpul yang ditunjuk oleh pointer Akhir akan menunjuk ke simpul pertama yang baru (yang ditunjuk oleh pointer Awal).

* **Kondisi *linked list* memiliki hanya satu simpul (data)**, seperti terlihat pada Gambar 3.15.



Gambar 3.15 Keadaan *list* hanya memiliki satu simpul

Ketika *circular linked list* hanya memiliki satu simpul, lalu ada proses penghapusan maka list akan kosong, dengan tahapan yang sama seperti di *single linked list* yang linear.

* **Kondisi *linked lis*t memiliki lebih dari satu simpul (data)**

Keadaan mula-mula *circular single linked list* (Gambar 3.16).



Gambar 3.16 Keadaan *list* sebelum terjadi proses penghapusan

Jika simpul pertama (yang ditunjuk oleh pointer awal) dihapus, maka *circular single linked list* menjadi seperti terlihat pada Gambar 3.17.



Gambar 3.17 Keadaan *list* setelah terjadi proses penghapusan

**3.b. Penghapusan di tengah**

Proses penghapusan di tengah sama dengan *single linked list* (lihat modul sebelumnya), karena tidak mengubah struktur dari *list*, tapi jika simpul yang akan dihapus ketemu di akhir, maka akan memanggil modul (prosedur) hapus di akhir/belakang.

**3.c. Penghapusan di belakang/akhir**

Penghapusan data di akhir adalah proses menghapus simpul terakhir (yang ditunjuk oleh variabel pointer Akhir), sehingga variabel pointer Akhir akan berpindah ke simpul sebelumnya, dan medan sambungan kanan (*next*) dari simpul yang ditunjuk oleh pointer Akhir yang baru akan menunjuk ke simpul pertama (yang ditunjuk pointer Awal).

* **Kondisi *linked list* memiliki hanya satu simpul (data)**

Sama seperti proses penghapusan di depan/awal.

* **Kondisi *linked list* memiliki lebih dari satu simpul (data)**

Keadaan mula-mula *circular single linked list* sebelum terjadi proses penghapusan (Gambar 3.18).



Gambar 3.18 Keadaan *list* sebelum terjadi proses penghapusan

Jika simpul terakhir (yang ditunjuk oleh pointer akhir) dihapus, maka *circular single linked list* menjadi seperti terlihat pada Gambar 3.19.



Gambar 3.19 Keadaan list setelah terjadi proses penghapusan

**III.2. *Circular Double Linked List***

Operasi-operasi yang ada pada *Circular Double Linked List* hampir sama seperti pada *Double Linked List* yang telah dibahas sebelumnya.

Operasi-operasinya sebagai berikut:

1. **Penciptaan**

Penciptaan adalah memberikan harga *nil* terhadap variabel pointer kepala (pointer Awal dan pointer Akhir), seperti terlihat pada Gambar 3.20.



Gambar 3.20 Hasil dari proses penciptaan *list*

1. **Penyisipan**

**2.a Penyisipan di depan/awal**

Operasi ini berguna untuk menambahkan satu simpul baru di posisi pertama. Langkah pertama untuk penambahan data adalah pembuatan simpul baru dan mengisinya dengan data pada *field* info-nya. Pointer yang menunjuk ke simpul tersebut dipanggil dengan nama **Baru**.

Ada 2 kondisi yang harus diperhatikan dalam penambahan data di depan/awal yaitu :

* 1. **Ketika linked list masih kosong**

Kalau kondisi *linked list* masih kosong, maka simpul baru akan menjadi simpul awal dan sekaligus simpul akhir dari *circular double linked list*.

* Kondisi sebelum disisipkan (Gambar 3.21)



Gambar 3.21 Keadaan list sebelum terjadi proses penyisipan

* Kondisi setelah operasi penyisipan

Operasi penyisipan pertama kali ketika *linked list* masih kosong adalah dengan mengisikan alamat pointer baru ke pointer Awal dan pointer Akhir. Dan memberikan nilai *nil* pada medan sambungan kiri (*prev*) dan medan sambungan kanan(*next*). Keadaan circular linked list setelah terjadi proses penyisipan (Gambar 3.22).



Gambar 3.22 Keadaan *list* setelah proses penyisipan

* 1. **Ketika *linked list* tidak kosong**

Misalnya mula-mula keadaan *list* sebelum terjadi proses penyisipan ada dua simpul, seperti terlihat pada Gambar 3.23.



Gambar 3.23 Keadaan list sebelum proses penyisipan

* Misalkan akan menyisipkan simpul baru berisi data 2



* + Setelah simpul yang berisi data 2 disisipkan di depan/awal, maka *circular double linked list* setelah proses penyisipan seperti Gambar 3.24.



Gambar 3.24 Keadaan list setelah proses penyisipan

**2.b. Penyisipan di tengah**

Proses penyisipan di tengah sama seperti pada *double linked list* (lihat modul sebelumnya), karena tidak mengubah struktur *circular*-nya.

Untuk proses tersebut ada dua hal yang harus diperhatikan yaitu :

* **Kondisi *linked list* masih kosong** prosesnya sama seperti penyisipan di depan/awal.
* **Kondisi *linked list* sudah mempunyai data (tidak kosong)**

Mula-mula keadaan *linked list* sebelum proses penyisipan (Gambar 3.25).

Gambar 3.25 Keadaan *list* sebelum proses penyisipan

Misalnya akan menyisipkan data 4 (Gambar 3.26) sebelum data 9 (untuk menyisipkan data setelahnya lihat kembali pada *single linked list*)



Gambar 3.26 Simpul yang akan disisipkan

Maka bentuk *circular double linked list* setelah mengalami penyisipan di tengah seperti terlihat pada Gambar 3.27.

Gambar 3.27 Keadaan list setelah proses penyisipan

**2.c. Penyisipan di belakang/akhir**

Operasi ini berguna untuk menambahkan simpul (data) baru di posisi akhir. Langkah pertama untuk penambahan data adalah pembuatan simpul baru dan pengisian nilai pada medan datanya. Pointer yang menunjuk ke simpul yang berisi data yang akan disipkan tersebut dipanggil dengan nama **Baru**.

Misalkan akan menyisipkan data 9 di akhir (Gambar 3.28).



Gambar 3.28 Simpul yang akan disisipkan

Ada 2 kondisi yang harus diperhatikan dalam penyisipan simpul (data) di akhir yaitu :

* **Kondisi *linked list* masih kosong** prosesnya sama seperti penyisipan di depan/awal.
* **Ketika *linked list* sudah mempunyai simpul (data)**

Mula-mula keadaan *list* sebelum terjadi proses penyisipan di belakang/akhir seperti terlihat pada Gambar 3.29.



Gambar 3.29 Keadaan list sebelum terjadi proses penyisipan

Setelah simpul (data) baru disisipkan di akhir, maka *circular double linked list* seperti terlihat pada Gambar 3.30.



Gambar 3.30 Keadaan *list* setelah terjadi proses penyisipan di belakang

1. **Penghapusan**
   1. **Penghapusan di awal**

Operasi ini berguna untuk menghapus simpul (data) di posisi pertama. Ada 3 keadaan yang mungkin terjadi ketika akan melakukan proses hapus yaitu :

* **Kondisi *linked list* masih kosong**

Jika kondisi ini terjadi, maka proses penghapusan data tidak bisa dilakukan karena *linked list* masih kosong.

* **Kondisi *linked list* hanya memiliki satu simpul (data)**

Langkah yang perlu dilakukan ketika ingin melakukan proses penghapusan *linked list* yang memiliki hanya satu simpul (data) adalah dengan cara menempatkan dahulu satu pointer bantuan (phapus), kemudian medan data dari simpul yang akan dihapus disimpan ke dalam sebuah variabel (elemen), kemudian pointer Awal dan Akhir diberi harga *nil* dan simpul yang ditunjuk oleh pointer phapus langsung dihapus.

Untuk lebih jelas perhatikan urutan penghapusannya di bawah ini :

* + - Kondisi *list* sebelum dihapus (Gambar 3.31)



Gambar 3.31 Keadaan *list* sebelum terjadi proses penghapusan

* + - Kondisi *list* setelah ada proses penghapusan (Gambar 3.32)

|  |  |  |
| --- | --- | --- |
|  | menjadi |  |

Gambar 3.32 Keadaan *list* setelah terjadi proses penghapusan

* **Kondisi linked list memiliki lebih dari satu simpul (data)**

Untuk operasi penghapusan data di posisi pertama pada *circular double linked list* yang mempunyai data lebih dari satu buah adalah :

Misalkan keadaan list mula-mula seperti terlihat pada Gambar 3.33.



Gambar 3.33 Keadaan *list* sebelum terjadi proses penghapusan

Setelah terjadi penghapusan di awal/depan, maka keadaan *list* seperti terlihat pada Gambar 3.34.



Gambar 3.34 Keadan *lis*t setelah terjadi proses penghapusan

* 1. **Penghapusan di tengah**

Proses penghapusan di tengah sama dengan *single linked list* (lihat modul sebelumnya), karena tidak mengubah struktur dari *list*, tapi kalau simpul yang akan dihapus ketemu di akhir, maka akan memanggil modul (prosedur) hapus di akhir/belakang.

* 1. **Penghapusan di akhir**

Penghapusan simpul (data) di akhir adalah proses menghapus simpul terakhir (yang ditunjuk oleh variabel pointer Akhir), sehingga variabel pointer Akhir akan berpindah ke simpul sebelumnya, dan medan sambungan kanan (*next*) dari simpul yang ditunjuk oleh pointer Akhir yang baru akan menunjuk ke simpul pertama (yang ditunjuk oleh pointer awal), dan medan sambungan kiri (prev) dari simpul yang ditunjuk oleh pointer awal menunjuk ke simpul terakhir (yang ditunjuk oleh pointer Akhir).

Ada 3 keadaan yang mungkin terjadi ketika akan melakukan proses hapus yaitu :

* + 1. **Kondisi *linked li*st hanya memiliki satu simpul (data)**

Penghapusan di akhir prosesnya sama seperti penghapusan di depan

* + 1. **Kondisi *linked list* memiliki lebih dari satu simpul (data)**

Misalkan mula-mula keadaan *list* sebelum terjadi proses penghapusan memiliki tiga simpul double (Gambar 3.35).



Gambar 3.35 Keadaan list sebelum terjadi proses penghapusan

Setelah terjadi proses penghapusan di akhir, maka keadaan *circular double linked list* seperti terlihat pada Gambar 3.36.



Gambar 3.36 Keadaan list setelah terjadi proses penghapusan

1. **Penelusuran/*Traversal***

Prosesnya secara umum sama seperti penelusuran pada *single liked li*st atau *double linked list* (lihat modul sebelumnya).

1. **Pencarian /*Searching***

Prosesnya secara umum sama seperti pencarian pada *single liked l*ist atau *double linked list* (lihat modul sebelumnya).

1. **Pengurutan/*Sorting***

Prosesnya secara umum sama seperti pengurutan pada *single liked list* atau *double linked list* (lihat modul sebelumnya).

1. **Penghancuran/*Destroy***

Proses penghancuran bisa dengan cara memanggil modul/subrutin penghapusan di awal atau penghapusan di akhir secara terus menerus sampai *list* kosong, atau dengan algoritma di bawah ini:

phapus 🡨 Awal

While (Awal ≠ nil) do

Awal 🡨 Awal↑.next

dealloc(phapus)

phapus 🡨 Awal

EndWhile

Akhir 🡨 nil