UAS

Mata Kuliah	:	Struktur Data
Dosen Pengampu	:	Andi Moch Januriana, ST., M.Kom

NAME : Mustopa

NIM : 3337220023

KELAS : C

Program Studi Informatika

Fakultas Teknik

Universitas Sultan Ageng Tirtayasa

1. **Kelebihan :**
    - double linked list bisa ke node sebelum dan sesudahnya sedangkan single linked list Hanya bisa Ke sesudahnya
      Sehingga lebih efisien dalam setiap operasi pada double linked list

    Kekurangan :
    - Penggunaan memory Double linked list Lebih besar di bandingkan single linked list
    - Implementasi yang rumit : double linked list juga memiliki implementasi yang rumit di bandingkan single linkedlist Setiap operasi nya harus lebih teliti dan memastikan keterhubungan antar node node nya benar .

2. Konsep antrian bank menggunakan double link list circular memiliki keuntungan dan pertimbangan
    Keuntungan :
    - Akses yang efisien : Dengan menggunakan double link list kita bisa dengan mudah mengakses elemen elemen dari dua arah baik maju atau mundur dan pemprosesan operasi penambahan , peenghapuan dll yang lebih cepat
    - Perulangan tanpa batas : dalam double linked list circular elemen terahir terhubung kembali ke elemen yang pertama membentuk lingkaran. ini memungkinkan antrian untuk terus berputar dan melayani pelanggan baru tanpa harus menutup antrian saat mencapai batas tertentu.

    Pertimbangan :

    - Implemntasi yang komplex : dengan menggunakan double linked list circular implementasi nya akan lebih komplex harus lebih teliti pengelolaan penambahan, penghapusan, dan perulangan antrian dilakukan dengan benar.
    - Penggunaan Memory

3.

- **Deklarasi**

```
struct AntrianBank
{
    string nama;

    AntrianBank *next;
    AntrianBank *prev;
};

AntrianBank *head, *tail, *bantu, *newNode, *del;
```

Disini saya membuat struck dengan nama antrian bank dengan atribut nama dana **varibel pointer** next dan prev

Variable pointer global head ,tail ,bantu ,newnode ,del

- **Membuat Double linked list circular**

```
void createDoublelinked(string nama)
{
    head = new AntrianBank();
    head->nama = nama;

    head->prev = head;
    head->next = head;
    tail = head;
}
```

Sebagai data awal kita jadikan dulu sebagai head dan tail

- **Menambahkan Pelanggan dari depan depan (dalam antrian bank ini bisa di sebut pelanggan VIP karena datang awal Kebagian nomor antrian paling depan)**

```
// tambah depan
void tambahpelanggandepan(string nama)
{
    if (head == NULL)
    {
        cout << "Double Linked List belum dibuat!!!";
    }
    else
    {
        newNode = new AntrianBank();
        newNode->nama = nama;

        newNode->prev = tail;
        newNode->next = head;
        head->prev = newNode;
        tail->next = newNode;
        head = newNode;
    }
}
```

newNode Prev nya menunujuk ke tail (karena circular)
newNode Next nya ke head (karena setelah newnode nya kita menunjuk ke head)
head prev nya ke newNode
tail next ke newnode
head = newnode (menjadikan Newnode sebagai head)

sehingga akan menambahkan di paling depan dan keterhubungan antar node akan benar

- **Menambahkan Pelangan dari Belakang (dalam antrian bank disebut Pelanggan Biasa)**

```
void tambahpelangganbelakang(string nama)
{
    if (head == NULL)
    {
        cout << "Double Linked List belum dibuat!!!";
    }
    else
    {
        newNode = new AntrianBank();
        newNode->nama = nama;

        newNode->prev = tail;
        newNode->next = head;
        tail->next = newNode;
        tail = newNode;
    }
}
```

Sama saja kaya Tambah depan hanya saja kita ubah keterhubungan antar node nya

Newnode prev nya kita tunjuk ke tail (karena kita tambah belakang tentu saja tail sebelumnya ada di sebelum newnode ini)
Newnode next nya ke head (karna circular)
Tail next nya ke newnode (karna setelah tail ada newnode setelahnya)
Tail = newnode (menjadikan newnode sebagai tail yang baru)

- **Hapus depan dan belakang**

```cpp
// remoevdepan
void hapusdepan()
{
    if (head == NULL)
    {
        cout << "Double Linked List belum dibuat!!!";
    }
    else
    {
        del = head;
        head = head->next;
        head->prev = tail;

        delete del;
    }
}
void hapusbelakang()
{
    if (head == NULL)
    {
        cout << "Double Linked List belum dibuat!!!";
    }
    else
    {
        del = tail;
        tail = tail->prev;
        tail->next = head;
        delete del;
    }
}

```

Hapus depan

Menggunakan variable pointer del dan dijadikan sebagai head (karena head posisinya kemungkinan paling depan)

Head = head -> next (menjadikan head-> next sebagai head yang baru)

Head->prev = tail (circular)

Kemudian tinggal delete varibel pointer del

Hapus belakang

Menggunakan variable pointer del dan dijadikan sebagai tail (karena tail posisinya kemungkinan paling belakang)

tail = tail -> prev (menjadikan tail-> prev sebagai tail yang baru)

tail->next = head (circular)

Kemudian tinggal delete varibel pointer del

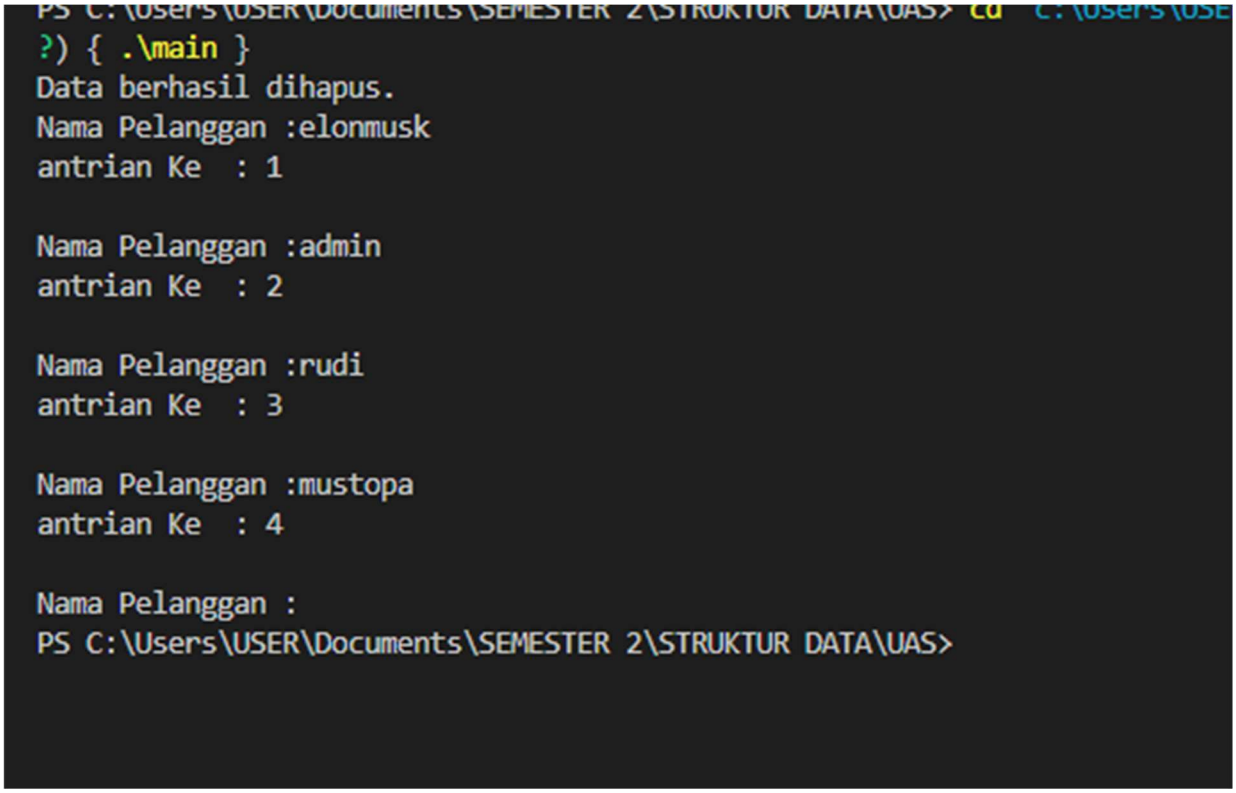
- **Hapus pelanggan By nama**

```
// menghapus node tertentu
void removeByname(string nama)
{
    if (head->nama == nama)
    {
        if (head == tail)
        {
            delete head;
            head = NULL;
            tail = NULL;
        }
        else
        {
            AntrianBank *temp = head;
            head = head->next;
            head->prev = tail;
            tail->next = head;
            delete temp;
        }
        cout << "Data berhasil dihapus." << endl;
        return;
    }
    bantu = head->next;
    while (bantu != head)
    {
        if (bantu->nama == nama)
        {
            bantu->prev->next = bantu->next;
            bantu->next->prev = bantu->prev;
            delete bantu;
            cout << "Data berhasil dihapus." << endl;
            return;
        }
        bantu = bantu->next;
    }
    cout << "Data tidak ditemukan." << endl;
}
```

- **Int main**

```
int main()
{

    createDoublelinked("admin");
    tambahpelangganbelakang("rudi");
    tambahpelangganbelakang("mustopa");
    tambahpelangganbelakang("cic");
    tambahpelangganbelakang("salwa");
    tambahpelanggandepan("elonmusk");
    tambahpelanggandepan("Putin");

    hapusbelakang();
    hapusdepan();
    removeByname("cic");

    printDoublelinkedlist();

    return 0;
}
```

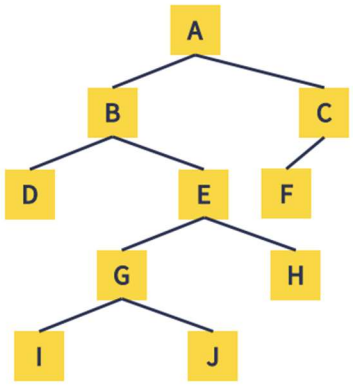
- **Output**



FULL KODE PROGRAM :

https://github.com/mustopa17/Strukturdata/tree/main/UAS_PROJECT_ANTRIAN_BANK_DOUBLELINKEDLISTCIRCULAR

4. **Tree yang akan di uraikan**



Pre order

1. Kunjungi root saat ini.
2. Telusuri subtree kiri.
3. Telusuri subtree kanan.

PreOrder = **A, B , D, E ,G ,I , J, H, C, F**

- Kunjungi rootnya **A**
- Telusuri subtree kiri **B , (** **B** masih punya subtree kiri **D** dan **D** tidak punya subtree kiri dan kanan maka kembali ke root **B**)
  hasil sementara =**A, B , D**

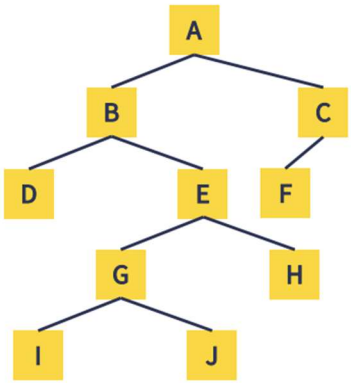
- Telusuri Subtree Kanan **B** punya subtree kanan **E**
  hasil sementara =**A, B , D, E**
- Telusuri subtree kiri **E (** **E** masih punya subtree kiri **G** dan **G** punya subtee kiri **I** dan **I** tidak punya subtree kiri dan kanan maka kembali ke root **G** dan **G** punya subtree kanan **J J** tidak punya subtree kanan dan kiri maka kembali ke root **E)**
  hasil sementara =**A, B , D, E ,G ,I , J**
- Telusuri Subtree Kanan **E** (**E** punya subtree kanan **H** dan **H** tidak punya subtree kiri dan kanan maka kembali ke root **A**)
  hasil sementara =**A, B , D, E ,G ,I , J, H**
- Telusuri subtree kanan **A** (**A** punya subtree kanan **C** dan **C** punya subtree kiri **F** dan **F** tidak punya subtree kiri dan kanan**)**
  Hasil ahir =**A, B , D, E ,G ,I , J, H, C, F**

Kesimpulan :

telusuri terlebih dahulu root saat ini apakah masih punya subtree kiri atau kanan jika punya dahulukan subtree kiri
Dan jika tidak ada subtrre kiri atau kanan maka kembali ke root awal

inOrder

1. Telusuri subtree kiri.
2. Kunjungi root nya.
3. Telusuri subtree kanan



inOrder = D,B,I,G,J,E,H,A,F,C

NULL = tidak punya subtree kanan dan kiri

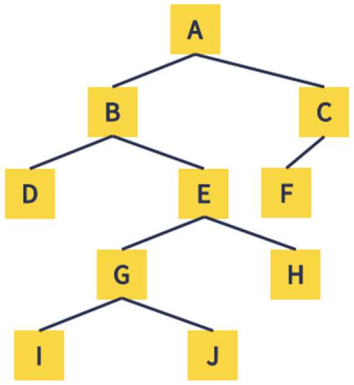
- Telusuri subtree kiri **D (D = NULL)**
- Kunjungi root **D= B**
- Telusuri subtree kanan **B = E (E** punya subtree kiri **G G** punya subtree kiri **I I = NULL**) hasil sementara = **D , B, I ,**
- Kunjungi root **I = G**
- Telusuri subtree kanan **G = J J= NULL (** kembali ke root **E) E** punya subtree kanan **H H=NULL (** kembali ke root **A)**
  hasil sementara = **D , B, I , G , J , E, H**
- Kunjungi root awal (karena kanan kiri sudah di telusuri) root awal= **A**
- Telusuri subtree kanan **A= C C** punya subtree kiri **F**
- Kunjungi root **F = C**
  hasil ahir = **D , B, I , G , J , E, H , A, F , C**

Kesimpulan :

telusuri terlebih dahulu subtree kiri saat ini apakah masih punya subtree kiri jika punya dahulukan subtree kiri kemudian ambil rootnya kunjungi subtree kanan cek lagi apakah punya subtree kiri jika punya dahulukan dst...

postOrder

1. Telusuri subtree kiri.
2. Telusuri subtree kanan.
3. Kunjungi root nya.



postOrder = D,I,J,G,H,E,B,F,C,A

NULL = tidak punya subtree kanan dan kiri

- Telusuri subtree kiri **D (D = NULL)**
- Telusuri subtree kanan **B = E (E** punya subtree kiri **G G** punya subtree kiri **I I = NULL**)
  hasil sementara = **D , I ,**

- Telusuri subtree kanan **G = J J= NULL**
- Kunjungi root **= G**
  hasil sementara = **D , I , J, G**

- Kembali ke root **E**
- Telusuri subtree kanan **E = H H= NULL**
  hasil sementara = **D , I , J, G, H**

- kunjungi root = **E , B**
  hasil sementara = **D , I , J, G, H, E, B**

- kembali ke root **A**

- Telusuri subtree kanan **A = C (C** punya subtree kiri **F F = NULL**)
  hasil sementara = **D , I , J, G, H, E, B** , **F, C**

- Kunjungi root = **A** hasil ahir = **D , I , J, G, H, E, B** , **F, C, A**

Kesimpulan :

telusuri terlebih dahulu subtree kiri saat ini apakah masih punya subtree kiri jika punya dahulukan subtree kiri kemudian ambil subtree kanan dan ambil rootnya