4 *LINKED LIST*

Linked list adalah struktur berupa rangkaian elemen saling berkaitan dimana tiap elemen dihubung elemen lain melalui *pointer*. *Pointer* adalah alamat elemen. Penggunaan *pointer* untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walau tidak bersebelahan secara fisik di memori.

4.1 Istilah-istilah

1. Simpul

   Simpul terdiri dari dua bagian, yaitu :

   - Bagian data
   - Bagian *pointer* yang menunjuk ke simpul berikutnya.

2. First

   Variable first berisi alamat/*pointer* menunjuk lokasi simpul pertama *linked list*, digunakan sebagai awal penelusuran *linked list*.

3. Nil atau null

   Tidak bernilai, untuk menyatakan tidak mengacu ke manapun.

4. Simpul terakhir

   Simpul terakhir *linked list* berarti tidak menunjuk simpul berikutnya. Tidak terdapat alamat disimpan di field *pointer* (bagian kedua dari simpul). Nilai null atau nil disimpan di field *pointer* pada simpul terakhir.

Maka

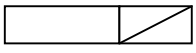
1) *Linked list* kosong adalah *linked list* dengan First = Nil
2) Elemen terakhir *linked list* dikenal dengan last^.next = Nil

Syarat *linked list* : Harus dapat diketahui alamat simpul pertama atau harus terdapat varibel Fisrt.

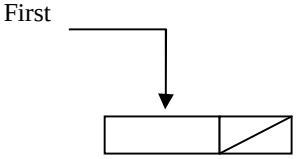
4.2 *Singly linked list*

Tipe *linked list* paling sederhana adalah *linked list* dengan simpul berisi satu link atau pointer mengacu ke simpul berikut. *Linked list* ini disebut *Singly linked list*.

a) Simpul *linked list*



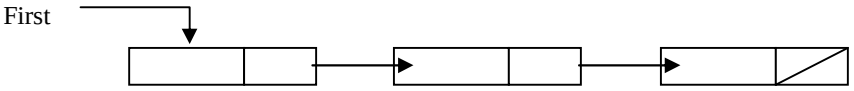
b) First



c) *Linked list* kosong

First = Nil

d) *Singly linked list* Tiga elemen



4.3 Deklarasi Struktur Data

```
Deklarasi 4.1
Type
         Error_type = (NoError, memoryFull);
         Data_type = …;
         Key_type = …;
         nodePtr = ^node;
         node = record
            key : key_type;
            data _ data_type;
            next : nodePtr;
               end;
linked list = record
            First : nodePtr;
            errorCode : error_type;
              end;
```

4.4 Operasi Dasar pada *Linked list*

Operasi paling mendasar yang dilakukan pada *linked list* adalah

1. Penciptaan dan penghancuran simpul (buildNode dan destroyNode)
2. Inisialisasi dan fungsi pemeriksaan *linked list* kosong (initL dan esEmptyL)
3. Penyisipan simpul ke *linked list*
   - Penyisipan sebagai simpul pertama (insertFirst)
   - Penyisipan setelah simpul tertentu (insertAfter)
   - Penyisipan sebagai simpul terakhir (insertLast)
   - Penyisipan sebelum simpul tertentu (insertBefore)
4. Penghapusan variable dinamis (simpul suatu *linked list*)
   - Penghapusan simpul pertama (deleteFirst)
   - Penghapusan simpul terakhir (deleteLast)
   - Penghapusan setelah simpul tertentu (deleteAfter)
   - Penghapusan simpul tertentu (deleteNode dan deleteKNode)
5. Traversal atau penelusuran seluruh simpul (traverseL)
6. Pencarian simpul tertentu (searchNode)

Operasi-operasi di atas merupakan operasi paling mendasar. Operasi ini dapat berdiri sendiri sebagai satu procedure tapi seringnya merupakan bagian operasi lain.

4.4.1 Penciptaan dan peghapusan simpul

Operasi penciptaan simpul untuk membentuk satu simpul

```
Program 4.1
Function buildNode(k:key_type; x:data_type) : nodePtr
Var
     P : nodePtr;
Begin
     New(p);
     P^.key := k;
     P^.data := x;
     P^.next := nil;
End;
Procedure destroyNode(var p : nodePtr);
Begin
     Dispose(p);
End;
```

Operasi destroyNode hanya mengganti dispose. Pada operasi-operasi lain dapat langsung menggunakan dispose.

4.4.2 Inisialisasi *Linked list*

```
Program 4.2
Procedure initL(var L : linked list);
Begin
     L.First := nil;
End;
```

Procedure ini menghasilkan *linked list* kosong ditunjuk L.First, inisialisasi ini penting bila *linked list* juga memuat informasi-informasi lain.

Untuk memeriksa *linked list* kosong adalah

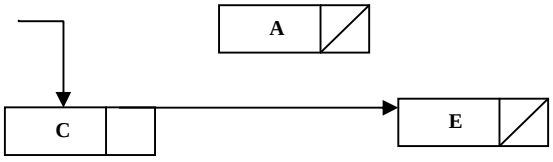
```
Program 4.3
Fuction isEmptyL(L : linked list) : Boolean;
Begin
      isEmptyL := (L.First = nil);
end;
```

4.4.3 Penyisipan simpul

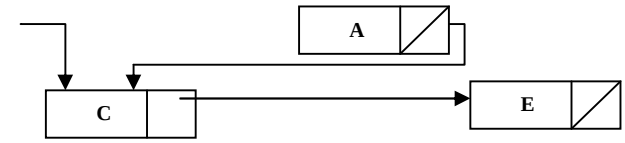
Penyisipan sebagai simpul pertama

Operasi ini akan menyisipkan elemen baru (berupa alamat) sebagai elemen pertama *linked list*. Skema penyisipan sebagai simpul pertama dapat digambarkan sebagai berikut:

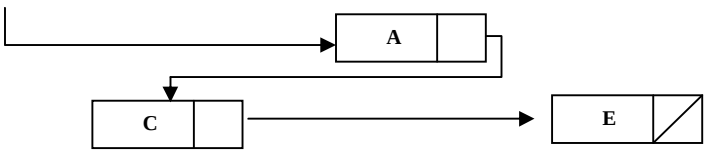
a) First



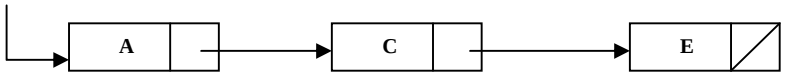
b) First



c) First



d) First




```
Program 4.4
Procedure   insertFirst(var   L   :   linked   list;
p:nodePtr);
Begin
      P^.next := L.First;
      L.First := p;
End;
```

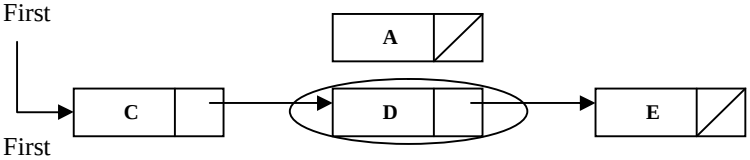
Langkah-langkah yang menjamin ketidakputusan rantai *linked list* adalah sebagai berikut:

1. Pointer next elemen baru menunjuk dulu elemen pertama dari *linked list*
2. L.First menunjuk ke elemen baru

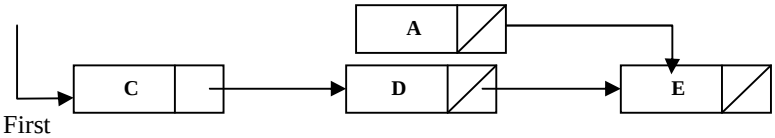
Penyisipan setelah simpul tertentu

Operasi ini akan menyisipkan elemen baru (berupa alamatnya) sebagai elemen setelah elemen tertentu di *linked list*. Skema penyisipan ini digambarkan sebagai berikut:

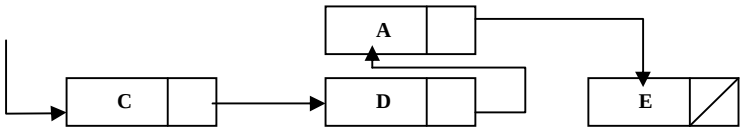
a) First



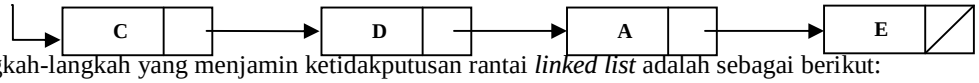
b) First



c) First



d) First



Langkah-langkah yang menjamin ketidakputusan rantai *linked list* adalah sebagai berikut:

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

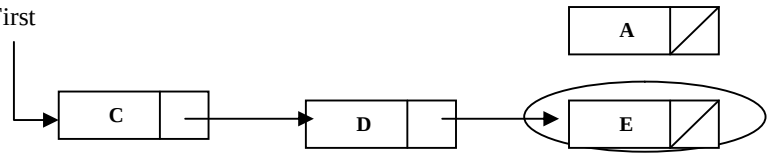
Procedure penyisipan ini dalam pascal adalah

```
Program 4.5
Procedure insertAfter(var prec : nodePtr; r:nodePtr);
Begin
            r^.next := prec^.next;
            prec^.next := r;
End;
```

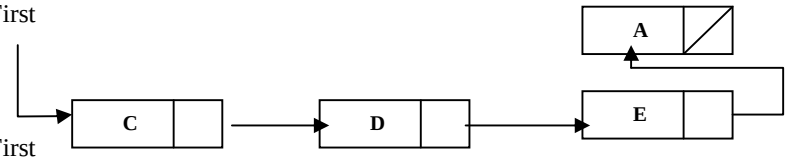
Penyisipan sebagai simpul terakhir

Operasi ini akan menyisipkan elemen baru (berupa alamatnya) sehingga elemen tersebut akan menjadi elemen terakhir *linked list*. Skema penyisipan ini digambarkan sebagai berikut:

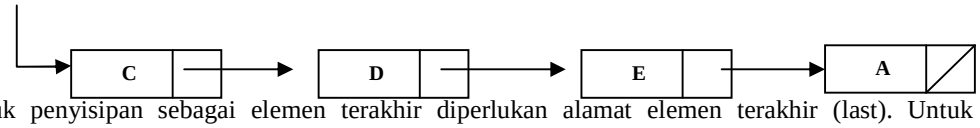
a) First



b) First



c) First



Untuk penyisipan sebagai elemen terakhir diperlukan alamat elemen terakhir (last). Untuk menemukan last diperlukan penelusuran sehingga terpenuhi last^.next = nil

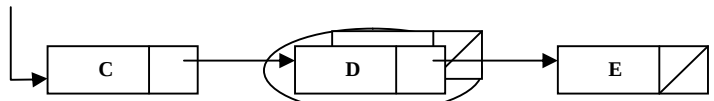
Langkah-langkah yang menjamin ketidakputusan rantai *linked list* adalah sebagai berikut:

1. Telusuri *linked list* sehingga mendapatkan elemen last
2. Lakukan insertaAfter setelah elemen last

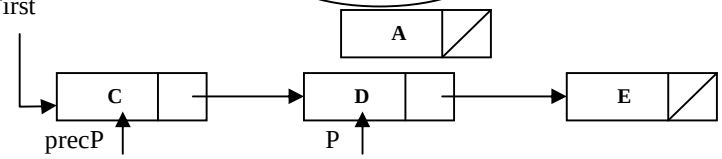
Penyisipan sebelum simpul tertentu

Operasi ini akan menyisipkan elemen baru (berupa alamatnya) sehingga elemen menjadi sebelum elemen tertentu. Bila tidak diketemukan maka tidak dilakukan penyisipan. Skema penyisipan ini digambarkan sebagai berikut:

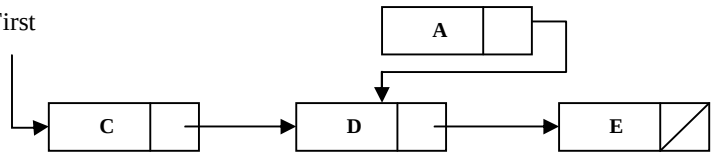
a) First



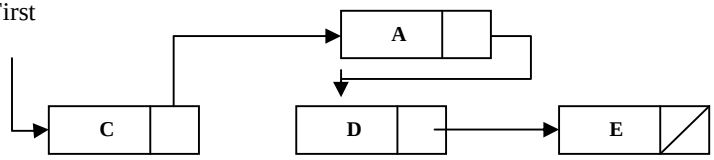
b) First



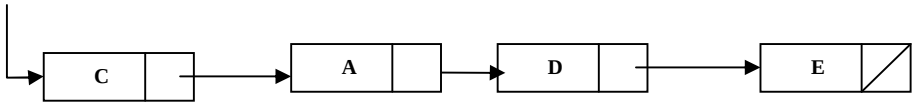
c) First



d) First



e) First



Langkah-langkah yang menjamin ketidakputusan rantai *linked list* adalah sebagai berikut:

1. Telusuri *linked list* sampai menemukan elemen tertentu sekaligus mencatat elemen sebelum elemen tertentu itu.
2. Procedure insertAfter setelah elemen sebelum elemen tertentu.

Program 4.7

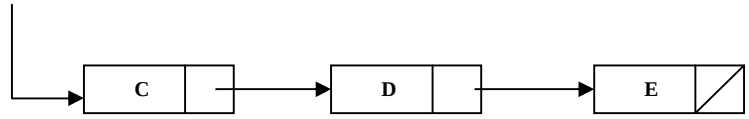
```
Procedure insertBefore (var L : linked list; k : key_type; x : data_type);
Var
      P, precP : nodePtr;
      Found : Boolean;
Begin
      If (L.First <> nil) then
            Begin
                  Found := false;
                  precP := nil;
                  P := L.First;
                  While (P <> nil) and (not found) do
                        If (P^.key = k) then
                              Found := true;
                        Else
                        Begin
                              precP := P;
                              P := P^.next;
                        End;
                  If found then
                  Begin
                        BuildNode (t, k, x);
                        T^.next := precP^.next;
                        precP^.next := t;
                  end;
            end;
end;
```

4.4.4 Penghapusan simpul

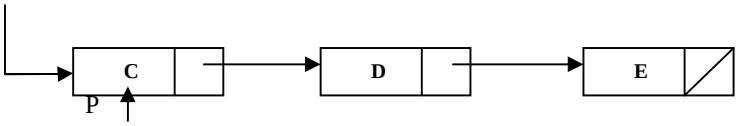
Penghapusan simpul pertama

Operasi ini akan menghapus elemen pertama *linked list*. Sebelum memanggil prosedur ini harus dijamin *linked list* tidak kosong. Skema penghapusan ini digambarkan sebagai berikut:

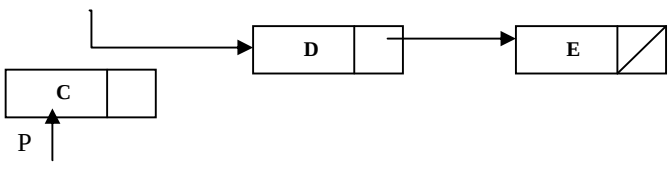
a) First



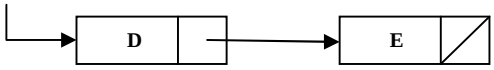
b) First



c) First



d) First



Langkah-langkah yang menjamin ketidakputusan rantai *linked list* adalah sebagai berikut:

1. Element First dicatat di suatu elemen
2. L.First mencatat elemen selanjutnya setelah L.First

Kedua langkah tersebut mengisolasi elemen pertama sebelumnya.

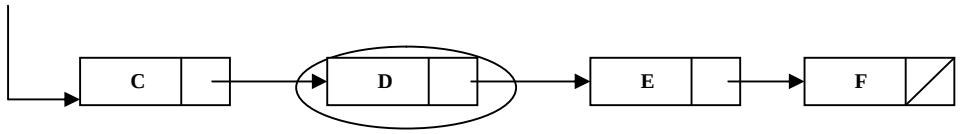
```
Program 4.8
Procedure deleteFirst(var L:linked list);
Var
      P : nodePtr;
Begin
      If (L.First <> nil) then
            Begin
                  P := L.First
                  L.First := L.First^.next;
                  Dispose(P);
            End;
End;
```

Prosedur Dispose merupakan prosedur standar di Pascal yang akan mengembalikan elemen dinamis ke memori sehingga dapat digunakan kembali.

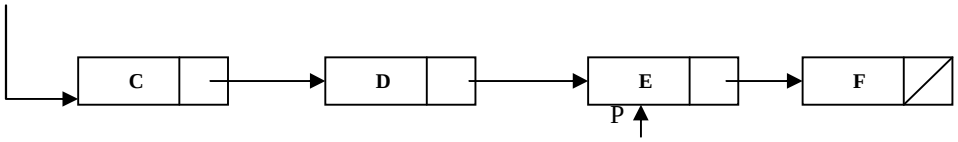
Penghapusan simpul setelah simpul tertentu

Operasi ini akan menghapus elemen setelah elemen tertentu. Skema penghapusan ini digambarkan sebagai berikut:

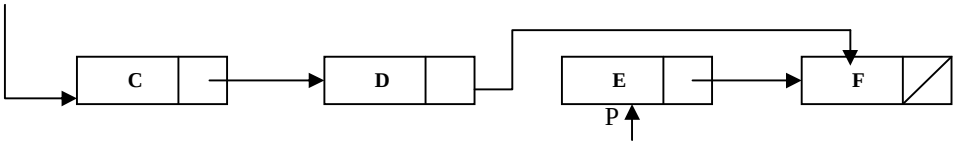
a) First



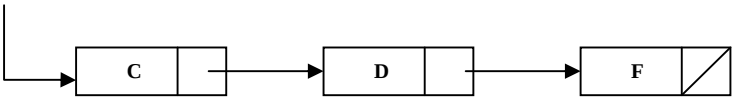
b) First



c) First



d) First



Langkah-langkah yang menjamin ketidakputusan rantai *linked list* adalah sebagai berikut:

1. Elemen tertentu yang dihapus dicat
2. Sambungkan ke elemen berikutnya

Kedua langkah tersebut mengisolasi elemen setelah elemen tertentu.

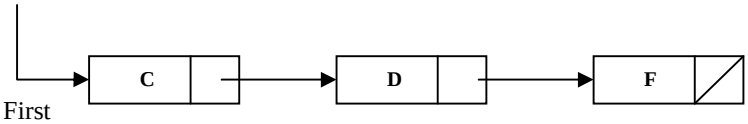
Program 4.9

```
Procedure deleteAfter(Var prec : nodePtr);
Var
      P : nodePtr;
Begin
      If (prec^.next <> nil) then
      Begin
            P := prec^.next;
            prec^.next := prec^.next.next;
            dispose(P);
      end;
end;
```

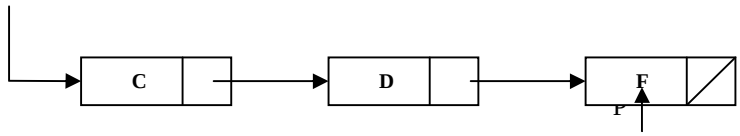
Penghapusan simpul terakhir

Operasi ini akan menghapus elemen pertama *linked list*. Sebelum memanggil prosedur ini harus dijamin *linked list* kosong. Skema penghapusan ini digambarkan sebagai berikut:

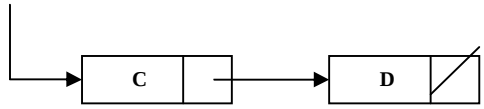
a) First



b) First



c) First



Program 4.10

```
Procedure deleteLast(Var L : linked list);
Var
      Last, precLast : nodePtr;
Begin
      If (l.First <> nil) then
      Begin
            Last := L.First
            precLast := Nil;
            while (Last^.next <> nil) then
            Begin
                  precLast := Last;
                  Last := Last^.next;
            End;
            P := Last;
            If (precLast = nil) then
```

```
                  L.First := nil
            Else
                  precLast^.next := nil;
                  Dispose(P);
      End;
End;
```

Penghapusan simpul tertentu

Operasi ini akan menghapus elemen tertentu. Terdapat dua versi penghapusan ini, yaitu :

- Elemen tertentu didasarkan pada alamat elemen
- Elemen tertentu didasarkan pada key

Untuk penghapusan elemen harus terlebih dulu diketahui elemen sebelum elemen tertentu tersebut dengan pencarian yang sekaligus mencatat elemen sebelumnya.

Langkah-langkah yang menjamin ketidakputusan rantai *linked list* adalah sebagai berikut:

1. Menelusuri apakah terdapat elemen tertentu tersebut sekaligus mencatat elemen sebelumnya
2. Setelah ditemukan maka hapus elemen tertentu dengan
   - Mengisolasi elemen tertentu
   - Menghubungkan elemen sebelumnya dengan elemen setelah elemen tertentu
   - Dispose elemen tertentu.

```
Program 4.11
Procedure deleteNode(Var L : linked list, Var P : nodePtr);
Var
      T, tPrec : nodePtr;
      Found : boolean;
Begin
      If (L.First <> nil) then
      Begin
            Found := false;
            T := L.First;
            tPrec := nil;
            While (T <> nil) and (not found) do
                  If (T = P) then
                        Found := true
                  Else
                  Begin
                        tPrec := T;
                        T := T^.next;
                  End;
            If (found) then
            Begin
                  If (tPrec = nil) then
                        L.First := T^.next;     {deletefirst}
                  Else
                        tPrec^.next := P^.next;
                        dispose(P);
            end;
      end;
end;
```

```
procedure deleteNode(Var L :linked list, k :key_type);
var
      t, tPrec : nodePtr;
      found : Boolean;
begin
      if (L.First <> nil) then
      begin
            found := false;
            t := L.First;
            tPrec := nil;
            While (t <> nil) and (notFound) do
                  If (t^.key = k) then
                        Found := true
                  Else
                  Begin
                        tPrec := t;
                        t := t^.next;
                  end;
            if (found) then
            begin
                  if (tPrec = nil ) then
                        L.First := t^.next;
                  Else
                  Begin
                        tPrec^.next := t^.next;
                  end;
                  dispose (t);
            end;
      end;
end;
```

4.4.5. Penelusuran seluruh simpul

Linked list adalah berisi sekumpulan elemen. Dalam beberapa operasi sering diperlukan pemrosesan semua elemen *linked list*. Berikut adalah dua versi penelurusan *linked list*.

1. Init adalah prosedur untuk persiapan pemrosesan
2. Visit (P) adalah prosedur pemrosesan simpul yang saat itu (dikunjungi atau ditunjuk)
3. Terminate adalah prosedur pemrosesan akhir setelah penelusuran

Program 4.12 **Versi 1 : menggunakan while Do…End**

```
Procedure ListTraversal(L : linked list);
Var
      P : nodePtr;
Begin
      Init;
      P := L.First;
      While (P <> nil ) do
      Begin
            Visit (P);
            P := P^.next;
      End;
      Terminate;
End;
```

Program 4.13 **Versi 2 : menggunakan Repeat…Until**

```
Procedure ListTraversal (L :linked list);
Var
      P : nodePtr;
Begin
      If (L.First = nil) then
            Message_linked list_Kosong;
      Else
      Begin
            Inisialisasi;
            P := L.First;
            Repeat
                  Proses(P);
                  P := P^.next;
            Until (P = nil);
            Terminasi;
      End;
End;
```

4.4.6. Pencarian simpul

Pencarian pada *linked list* hanya dapat dilakukan secara sekuen (berturutan). Pencarian merupakan modifikasi dari penelurusan dengan berhenti pada simpul tertentu.

Program 4.14

```
Function ListSearch(L : linked list, k : key_type, var P : nodePtr) :
Boolean;
Var
      Found : Boolean;
Begin
      P := L.First;
      Found := false;
      While (P <> nil) and (notFound) do
      Begin
            If (P^.key = key) then found := true
            else
                  P := P^.next;
      End;
      ListSearch := found;
End;
```

Fungsi mengirim true bila ditemukan dan alamat ditemukan dicatat pada argument p.

Kesimpulan

Bila operasi yang melibatkan pencarian sering dilakukan maka *singly linked list* harus dimodifikasi sehingga mempercepat proses pencarian. Modifikasi untuk mempercepat proses antara lain:

1. Menambah *sentinel* pada *singly linked list*. *Sentinel* adalah simpul *linked list* yang tidak dihitung sebagai elemen *linked list* hanya digunakan sebagai pemercepat
2. Mengelola *singly linked list* sebagai sorted *linked list* dimana elemen-elemen di *linked list* diurutkan menurut field tertentu menaik atau menurun.

4.5 Operasi Terhadap *Linked list*

Berikut adalah operasi-operasi yang sering diterapkan pada *linked list*, yaitu :

1. Penghapusan *linked list*
2. Inverse
3. Penyambungan
4. Panjang *linked list*

4.5.1. Penghapusan *linked list*

Operasi penghapusan *linked list* merupakan modifikasi penelusuran *linked list* dengan mencatat (mengisolasi lebih dulu) elemen yang akan dihapus.

Program 4.15

```
Procedure purgeL(var L : linked list);
Var
      P, temp : nodePtr;
Begin
      If (L.First <> nil) then
      Begin
            P := L.First;
            While (P <> nil) do
            Begin
                  Temp := P;
                  P := P^.next;
                  Dispose(temp);
            End;
      End;
End;
```

4.5.2. Inverse *linked list*

Operasi membalik *linked list* sehingga yang semula elemen terakhir menjadi elemen pertama dan sebaliknya.

Program 4.16

```
Procedure invertL(var L :linked list);
Var
      p,q,r : nodePtr;
begin
      p:=L.First;
      q:=nil;
      while (p <> nil) do
      begin
            r := q;
            q := p;
            p := p^.next;
            q^.next := r;
      end;
      L.First := q;
End;
```

4.5.3. Penyambungan dua *linked list*

Operasi menyambungkan dua *linked list* sehingga L3 akan berisi L1 disambung L2.

Program 4.17

```
Procedure ConcatL(Var L3: Linked list, L1,L2 : linked list);
Var
      Last1 : nodePtr;
Begin
      If (L1.First = nil) then
            L3.First := L2.First;
      Else
      Begin
            L3.First := L1.First;
            Last1 := L1.First;
            While (Last^.next <> nil) do
                  Last1 := Last1^.next;
                  Last1^.next := L2.First;
      End;
End;
```

4.5.4. Panjang *linked list*

Fungsi ini mengirim jumlah elemen di *linked list*.

Program 4.18

```
Function lengthL(L : linked list) : integer;
Var
      P : linked list;
Begin
      lengthL := 0;
      if (L.First <> nil) then
      begin
            P := L.First;
            Repeat
                  LengthL := LengthL + 1;
                  P := P^.next;
            Until (P = nil);
      End;
End;
```

5 *STACK*

Stack adalah kasus khusus ordered list dengan penyisipan dan penghapusan dilakukan di salah satu ujung. Misalnya *stack* S = (a_1, a_2, ….a_n), maka :

1. Elemen a_1 adalah elemen terbawah
2. Elemen a_i adalah elemen diatas elemen a_i-1, dimana $1<i\leq n$.

Batasan-batasan terhadap *stack* itu berimplikasi jika lemen A, B, C, D, E ditambahkan ke *stack*, maka elemen yang pertama dihapus adalah elemen terakhir yang dimasukan. *Stack* sering disebut **LIFO** (*Last In, First Out*) yaitu elemen yang lebih akhir disisipkan menjadi elemen yang paling dulu diambil. *Stack* juga disebut *pushdown list*.

5.1 Spesifikasi *Stack*

Berikut adalah struktur aksiomatik *stack*

Spesifikasi aksiomatik 5.1	
1	Structure *STACK*(item)
2	Declare
3	InitS() → *stack*
4	Push(item,*stack*) → *stack*
5	Pop(*stack*) → *stack*
6	Top(*stack*) → item
7	IsEmptyS(*stack*) → Boolean
8	End declare
9	For all S ∈ *stack*, i ∈ item, let
10	IsEmptyS(InitS) ::= true
11	IsEmpryS(Push(i,s)) ::= false
12	Pop(initS) ::= error
13	Pop(Push(i,S)) ::= S
14	Top(InitS) ::= error
15	Top(Push(i,S)) ::= i
16	End Structure

Representasi *stack* dapat dilakukan dengan menggunakan representasi kontingu (array) atau representasi dinamis (*linked list*). Kedua representasi mempunyai keunggulan dan kelemahan. Pemilihan representasi bergantung karkateristik aplikasi.

5.2 Representasi sekuen

Representasi stack paling sederhana adalah menggunakan array satu dimensi. Elemen pertama disimpan di array SElemen[1], elemen kedua di SElemen[2] dan elemen ke-I di SElemen[i]. diasosiasikan dengan array tersebut variable TOP menunjuk elemen puncak stack pada satu saat.

5.2.1. Deklarasi Stuktur Data

Deklarasi stack di pascal

```
Deklarasi 5.1
Const N = …; { diisi jumlah elemen stack maksimum}
Type
      errorType = (NoError, OverFlow, UnderFlow, NotAvailable);
      data_type = …; {diisi type data dari elemen-elemen di stack}
      stack = record
            top : integer;
            SElemen : array[1…N] of data_type;
            errorCode : errorType;
      end;
```

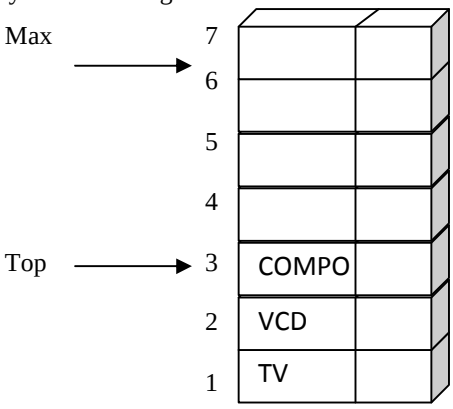
5.2.2. Operasi pada Stack

Operasi-operasi dasar pada stack

1. Operasi menciptakan S sebagai stack kosong, (initS(S))
2. Operasi menyisipkan elemen x ke stack S dan mengembalikan stack baru, (push(S,x))
3. Operasi menghilangkan elemen puncak stack S, (pop(S,x))
4. Operasi mengirim elemen puncak stack S, (tops (S,x))
5. Operasi mengirim True jika S kosong, jika tidak kosong mengirim False, (isEmptyS (S))
6. Operasi mengirim jumlah elemen stack S, (howManyInS (S))

5.2.3. Single stack dengan array

Sesuai dengan sifat stack, maka pengambilan/penghapusan elemen dalam stack harus dimulai dari elemen teratas. Ilustrasinya adalah sebagai berikut:



Di atas, tampak jelas visualisasi stack dengan array, dimana array ke-1 diisi oleh TV, indeks ke-2 diisi oleh VCD dan seterusnya. Top dianggap sebagai puncak dari Stack. Harus diingat bahwa bayak stack yang mungkin ada harus dibatasi karena alokasi memori pada array bersifat statis dan terbatas. Berikut contoh deklarasi konstanta, tipe dan variable yang akan dipakai dalam penjelasasn operasi-operasi pada stack dengan array.

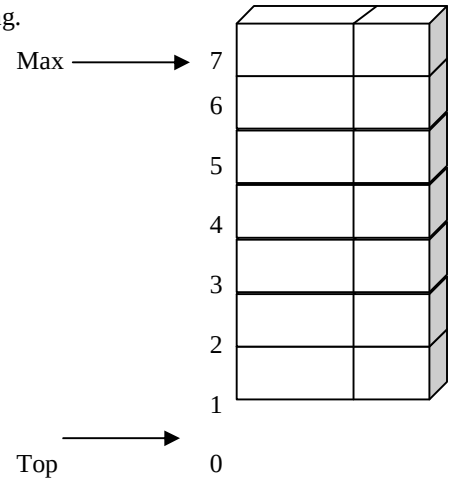
```
Const
      Max = 7;
Type  TipeData = byte;
      Stack = array [1..Max] of TipeData;
Var
      Top := TipeData
```

Operasi-operasi pada Single Stack dengan Array

1) Create : Membuat sebuah stack baru yang masih kosong.

```
Procedure Create;
Begin
      Top := 0;
End;
```

Konsepnya adalah bahwa Top menunjukan tingginya tumpukan stack. Jika tinggi tumpukan stack adalah 0, berarti tumpukan kosong.



2) Full : Function untuk memeriksa apakah stack yang ada sudah penuh.

```
Function Full : Boolean;
Begin
      Full := false;
      If (Top = Max) then
            Full := true;
End;
```

3) Push : menambahkan sebuah elemen kedalam stack. Tak bisa dilakukan jika stack sudah penuh.

```
Procedure Push (elemen : TipeData);
Begin
      If ( not full ) then
            Begin
                  Top := Top + 1;
                  Stack[Top]        :=
elemen;
            End;
End;
```

```
Procedure Push (elemen : TipeData);
Begin
      If ( not full ) then
            Begin
                  Top := 0 + 1;
                  Stack[1] := 29;
            End;
End;
```

4) Empty : Function untuk menentukan apakah stack kosong atau tidak

```
Function Empty : Boolean;
Begin
      Empty := False;
```

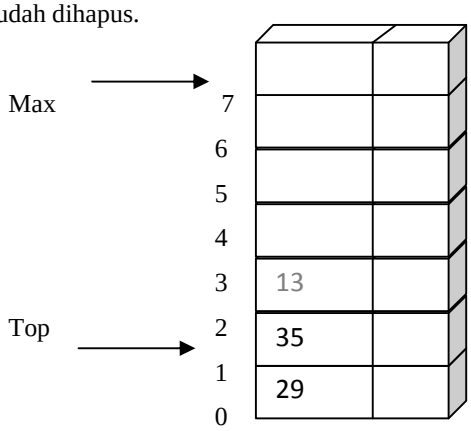
```
      If (Top = 0) then
            Empty := True;
End;
```

5) Pop : Mengambil elemen teratas dari stack. Stack tidak boleh kosong

```
Procedure Pop(Elemen : TipeData);
Begin
      If (not Empty) then
            Begin
                  Elemen := Stack[Top];
                  Top := Top – 1;
            End;
End;
```

```
Procedure Pop(Elemen : TipeData);
Begin
      If (not Empty) then
            Begin
                  Elemen := 13;
                  Top := 3 – 1;
            End;
End;
```

Jika posisi Top sudah berada dibawah sebuah elemen, maka elemen tersebut dianggap sudah hilang, jadi dalam kasus diatas 13 sudah dihapus.



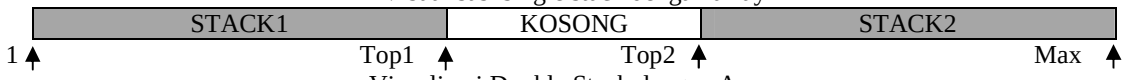
6) Clear : Mengosongkan stack (ket : jika Top = 0, stack dianggap kosong)

```
Procedure Clear;
Begin
      Top := 0;
End;
```

5.2.4. Double stack dengan array

Adalah teknik khusus yang dikembangkan untuk menghemat pemakaian memori dalam pembuatan dua stack dengan array. Intinya adalah penggunaan hanya sebuah array untuk menampung dua stack. Untuk mudahnya lihat gambar dibawah ini :

Visualisasi single stack dengan array



Visualisasi Double Stack dengan Array

Tampak jelas bahwa sebuah array dibagi untuk dua stack, stack1 bergerak ke kanan dan stack2 bergerak ke kiri. Jika Top1 (elemen teratas dari stack1) bertemu dengan Top2(elemen teratas dari stack2) maka double stack telah penuh. Berikut diberikan contoh deklarasi konstanta, tipe dan variable yang akan dipakai dalam penjelasan operasi-operasi double stak dengan array.

```
Const
      Max = 8;
```

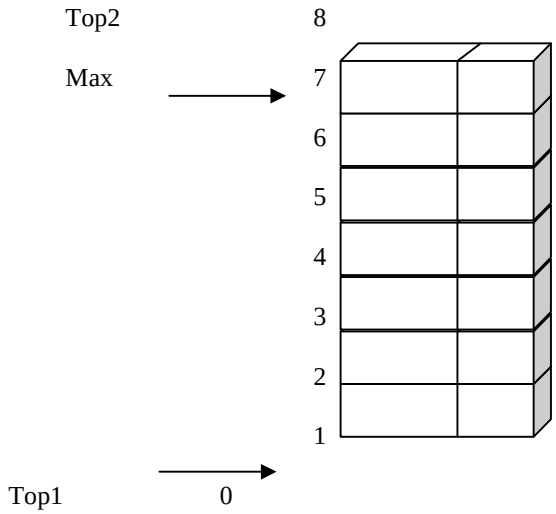
```
Type
      TipeData = Integer;
      Stack = array [1..Max] of byte;
Var
      Top := array [1..2] of byte;
```

Implementasi double stack dengan array dalam pascal adalah dengan memanfaatkan operasi-operasi yang tidak berbeda jauh dengan operasi single stack dengan array.

Operasi-operasi pada Double Stack dengan Array

1)Create: Membuat stack baru yang masih kosong

```
Procedure Create;
Begin
      Top[1] := 0;  {stack1 dianggap kosong jika Top[1] = 0}
      Top[2] := Max   { stack2 dianggap kosong jika Top[2] = Max+1}
End;
```



2)Full : Memeriksa apakah double stack sudah penuh

```
Function Full : Boolean;
Begin
      Full := False;
      If (Top[1]+1 >= Top[2]) then
            Full := True;
End;
```

3)Push : Memasukan sebuah elemen ke salah satu stack

```
Procedure Push(elemen : TipeData, NoStack : byte)
Begin
      If (not Full) then
      Begin
            Case NoStack of
                  1 : Top[1] := Top[1] +1;
                  2 : Top[2] := Top[2] – 0;
            End;
            Stack [Top[NoStack]] := elemen;
      End;
End;
```

```
Not Full then
Begin
      Top[1] := 0 +1;
       {NoStack=1}
      Stack[1] := 23;
      End;
```

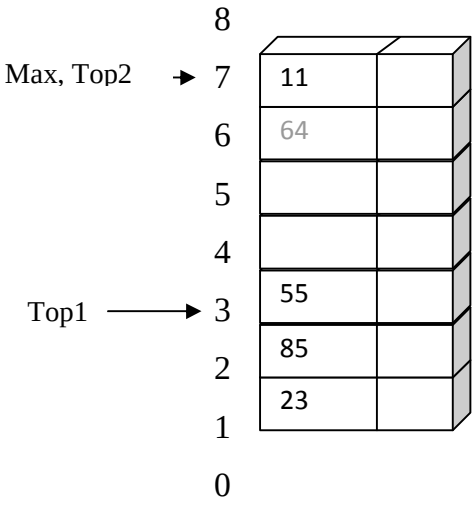
4)Empty : Function yang memeriksa apakah stack1 atau stack2 kosong.

```
Function Empty(NoStack : byte):Boolean;
Begin
      Empty := False
      Case NoStack of
            1 : if (Top[1] = 0) then
                  Empty := True;
            2 : if (Top[2] = Max + 1) then
                  Empty := True;
      End;
End;
```

5)Pop : Mengeluarkan elemen teratas dari salah satu stack

```
Procedure Pop(var elemen : TipeData, NoStack : byte)
Begin
      If (not Empty(NoStack)) then
      Begin
            Elemen := Stack[Top[NoStack]];
            Case NoStack of
                  1 : Top[1] := Top[1] -1;
                  2 : Top[2] := Top[2] + 1;
      End;
End;
```

```
Contoh : Pop(64,2)
not Empty=True then
Begin
      Elemen := 64;
      Top[2] := 6+1;    {NoStack = 2}
End;
```



6) Clear : Mengosongkan salah satu stack

```
Procedure Clear (NoStack : byte);
Begin
   Case NoStack of
      1 : Top[1] := 0;
      2 : Top[2] := Max + 1;
   End;
End;
```

5.2.5. Stack dengan Single linked list

Keunggulan stack menggunakan single linked list adalah penggunaan memori yang dinamis sehingga menghindari pemborosan memori. Misalnya dengan stack dengan array disediakan tempat untuk stack berisi 150 elemen, sementara ketika dipakai oleh user hanya diisi 50 elemen, maka telah terjadi pemborosan memori untuk sisa 100 tempat elemen yang tak terpakai. Dengan menggunakan linked list maka tempat yang disediakan akan sesuai dengan banyaknya elemen yang mengisi stack. Karena itu pula dalam stack dengan linked list tidak ada istilah **Full**, sebab biasanya program tak menentukan jumlah elemen stack yang mungkin ada (kecuali sudah dibatasi oleh pembuatnya).

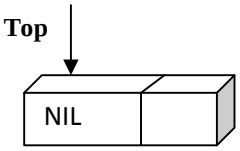
Berikut deklarasi tipe dan variable yang akan digunakan dalam penjelasan operasi stack dengan linked list.

```
Type
      TipeData = Byte;
      Point = ^Simpul;
      Simpul = record
                  Isi : TipeData;
                  Next : Point;
               End;
Var
      Top : Point;
```

Operasi-operasi pada stack dengan single linked list

1) Create : procedure untuk membuat stack baru yang kosong

```
Procedure Create;
Begin
      Top := nil;
End;
```



2) Empty : Function untuk memeriksa apakah stack yang ada masih kosong

```
Function Empy : Boolean;
Begin
      Empty := False;
      If (Top= null) then
            Empty := True;
End;
```

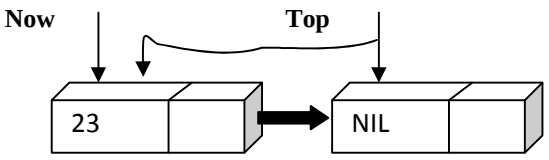
3) Push : Memasukan elemen baru kedalam stack

Push disini mirip dengan insert single linked list biasa.

```
Procedure Push (elemen : TipeData);
Var
      Now : Point;
Begin
      New(Now);
      Now.^isi := elemen;
      If (Empty) then
            Now.^next := nil;
      Else
            Now.^next := Top;
      Top := Now;
End;
```

Contoh PUSH(23)

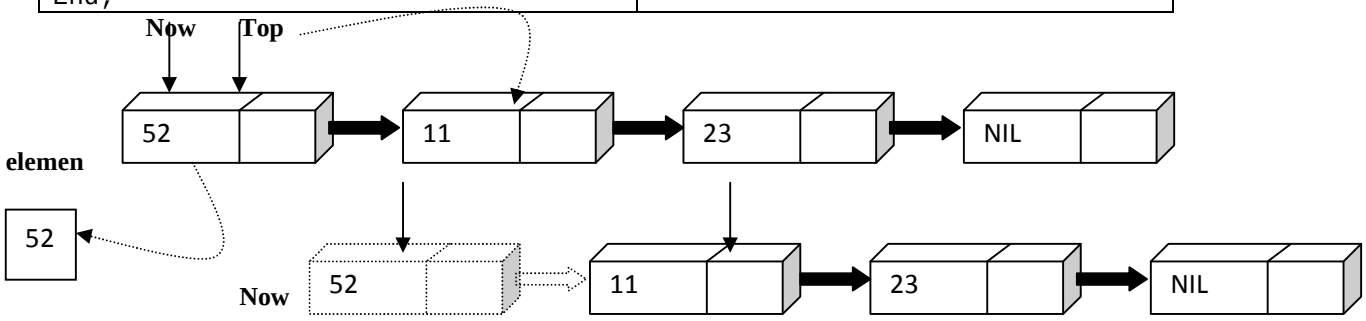
```
New(Now);
      Now.^isi := 23;
      Now.^next := nil;
             {empty=True)
      Top := Now;
```



4) Pop : Mengeluarkan elemen teratas dari stack

```
Procedue Pop(Var elemen : TipeData);
Var
      Now : Point;
Begin
      If (not Empty) then
      Begin
            Elemen := Now.^isi;
            Now := Top;
            Top := Top.^next;
            Dispose(Now);
      End;
End;
```

```
Contoh Pop(52)
Empty = False
      Elemen := 52;
      Now := Top;
      Top := Top.^next;
      Dispose(Now);
```



5) Clear : Menghapus stack yang ada.

```
Procedure Clear;
Var
      Trash : TipeData;
Begin
      While (not Empty) do
            Pop(Trash);
End;
```

```
{Empty = false}
elemen := 23;
Now := Top;
Top := Top.^next;
Dispose(Now);
```

Contoh Kasus

Buat program animasi yang menggambarkan sebuah piring diletakan jika ditekan angka 1. jika ditekan angka 1 lagi, diletakan piring baru di atas piring sebelumnya. Begitu terus sampai tumpukan piring maksimal 10 buah. Jika ditekan angka 2, piring teratas akan diangkat.

```
Uses crt;
Const
      Max = 10;
Type
      elemen_type = byte;
      Point = ^rec;
      rec = record
                  isi : elemen_type;
                  next : Point;
             end;
var
      Ch : char;
      Top : byte;
      Head : Point;

      Procedure PUSH(e :Elemen_type);
```

```
	Var
		Now : Point;
	Begin
		New(Now);
		Now.^isi := e;
		If (Head = nil) then
			Now.^next := nil
		Else
			Now.^next := Head;
		Head := Now;
	End;

	Procedure POP (e : elemen_type);
	Var
		Now : Point;
	Begin
		If (Head <> nil) then
		Begin
			e := Head.^isi;
			Now := Head;
			Head := Head.^next;
			Dispose(Now);
		End;
	End;

	Procedure PushAnim;
	Var
		i, y : byte;
	begin
		y := 0;
		Repeat
			Inc(y);
			GotoXY(32+top, y) write('|');
			For (i:= 1) to (30-2*top) do
			Begin
				GotoXY(32+top+I,y); write('_');
			End;
			GotoXY(62-top,y); write('|'); delay(15);
			If (y<>20-top) then
			Begin
				GotoXY(32,y); ClrEol;
			End;
		Until (y=20-top);
	End;

	Procedure PopAnim;
	Var
		i, y : byte;
	begin
		y := 20-top;
		repeat
			GotoXY(32,y); ClrEol;
			GotoXY(32+top,y); write('|');
			For (i := 1) to (30-2*top) do
			Begin
				GotoXY(32+top+i, y); write('_');
			End;
```

```
                  GotoXY(62-top, y); write('|'); delay(15);
                  GotoXY(32, y); ClrEol;
                  Dec(y);
            Until (y=1);
      End;

Begin
      Clrscr;
      Top := 0;
      Head := nil;
      Repeat
            GotoXY(1,1); write('Plane Simulation');
            GotoXY(1,2); write('================');
            GotoXY(1,3); write('1. Push Plate to Stack');
            GotoXY(1,4); write('2. Pop Plate from Stack');
            GotoXY(1,5); write('3. Exit');
            If (top <= Max) then
            Begin
                  GotoXY(1,9); ClrEol; write('Stack Number = ',top);
            End;
            Repeat
                  GotoXY(1,7); ClrEol; write('Your Choise [1/2/3 : ');
                  Ch := ReadKey; write(ch);
            Until (ch in['1'..'3']);
            Case ch of
                  '1' : begin
                         If (top < max) then
                         Begin
                               Inc(top);
                               Push(top);
                               PushAnim;
                        End else
                        Begin
                        GotoXY(3,13); write('Stack telah Penuh'); delay(80);
                               GotoXY(3,13); write('                ');
                        End;
                        End;
                  '2' : begin
                         If (top >= 1) then
                         Begin
                               POP(top);
                               PopAnim;
                               Dec(top);
                        End else
                        Begin
                        GotoXY(3,13); write('Stack telah kosong'); delay(80);
                               GotoXY(3,13); write('         ');
                        End;
                        End;
            End;
      Until ch in ['3'];
End;
```

6 QUEUE (ANTRIAN)

Queue jika diartikan secara harfiah berarti antrian. Queue merupakan salah satu contoh aplikasi dari pembuatan Double linked list yang cukup sering kita temui dalam kehidupan sehari-hari, misalnya pada saat mengantri di loket untuk membeli tiket.

Istilah yang sering cukup sering dipakai apabila seseorang masuk kedalam antrian adalah ENQUEUE. Dalam suatu antrian yang datang terlebih dahulu akan dilayani lebih dahulu.

Istilah yang sering dipakai bila seseorang keluar dari antrian adalah DEQUEUE. Untuk membuat aplikasi antrian dapat menggunakan metoda :

1) Array

a. Linier Array

b. Circular Array

2) linked list

Implementasi QUEUE dengan ARRAY

a. Linier Array

Linier array adalah suatu array yang dibuat seakan-akan merupakan suatu garis lurus dengan satu pintu masuk dan satu pintu keluar.

Berikut diberikan penggalan konstanta, type dan variable yang akan dipakai untuk menjelaskan , operasi-operasi dalam queue linier array. Dalam prakteknya, anda dapat menggantinya sesuai dengan kebutuhan anda.

```
Const
      MaxQueue = 6;
Type
      TypeQueue = byte;
Var
      Queue : Array [1..MaxQueue] of TypeQueue;
      Head, Tail : byte;
```

Operasi-operasi pada QUEUE dengan Linear Array :

1. Create

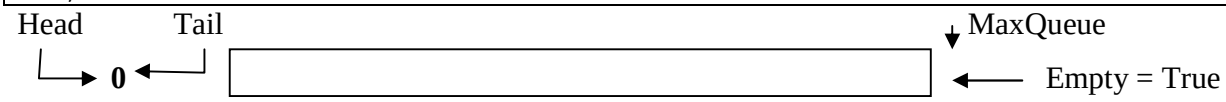
Procedure Create berguna untuk menciptakan QUEUE yang baru dan kosong yaitu dengan cara memberikan nilai awal (Head) dan nilai akhir (Tail) dengan nol (0). Nol menunjukan bahwa Queue (antrian) masih kosong. Berikut penggalan procedure create.

```
Procedure Create;
Begin
      Head := 0; Tail := 0;
End;
```

2. Empty

Function Empty berguna untuk mengecek apakah Queue masih kosong atau sudah berisi data. Hal ini dilakukan dengan mengecek apakah tail bernilai nol atau tidak, jika ya maka kosong. Berikut penggalan function empty :

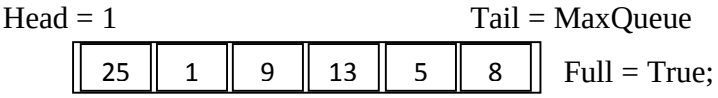
```
Function Empy : Boolean;
Begin
      If (Tail = 0) then
            Empty := True
      Else
            Empty := False;
End;
```



3. Full

Function Full berguna untuk mengecek apakah Queue sudah penuh atau masih bisa menampung data dengan cara mengecek apakah tail sudah sama dengan jumlah maksimal queue, jika ya maka penuh. Berikut penggalan function full :

```
Function Full : Boolean;
Begin
      If (Tail = MaxQueue) then
            Full := True
      Else
            Full := False
End;
```



4. EnQueue

Procedure EnQueue berguna untuk memasukan 1 elemen kedalam QUEUE. Berikut penggalan procedure EnQueue :

```
Procedure EnQueue(Elemen : Byte);
Begin
      If  (Empty) then
      Begin
            Head := 1; Tail := 1;
            Queue[Head] ::= Elemen;
      End Else
      If (Not Full) then
      Begin
            Inc (Tail);
            Queue(Tail) := Elemen;
      End;
End;
```

5. DeQueue

Procedure EnQueue berguna untuk mengambil 1 elemen dari QUEUE, operasi ini sering disebut juga SERVE. Hal ini dilakukan dengan cara memindahkan semua elemen satu langkah ke posisi di depannya, sehingga otomatis elemen yang paling depan akan tertimpa dengan elemen yang terletak di belakangnya, berikut penggalan procedure DeQueue

```
Procedure DeQueue;
Var
      i : Byte;
begin
      if (Not Empty) then
      Begin
            For (i := Head) To (Tail-1) Do
                  Queue[i] := Queue[i + 1];
                  Dec(Tail);
      End;
End;
```

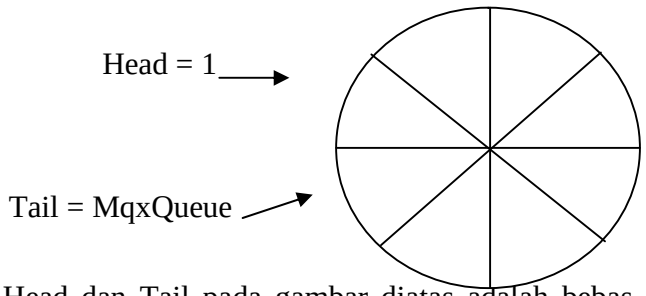
6. Clear

Procedure Clear berguna untuk menghapus semua elemen dalam QUEUE dengan jalan mengeluarkan semua elemen tersebut satu per satu sampai kosong dengan memanfaatkan procedure DeQueue, berikut penggalan procedure Clear.

```
Procedure Clear;
Begin
      While (Not Empty) Then
            DeQueue;
End;
```

b. Circular Array

Circular array adalah suatu array yang dibuat seakan-akan merupakan sebuah lingkaran dengan titik awal (Head) dan titik akhir (Tail) salang bersebelahan jika array tersebut masih kosong. Untuk lebih jelasnya perhatikan gambar berikut :



Posisi Head dan Tail pada gambar diatas adalah bebas asalkan saling bersebelahan. Operasi-operasi yang terdapat pada circular array tidak berbeda jauh dengan operasi pada linear array.

Operasi-operasi QUEUE dengan Circular Array

1. Create

Procedure Create berguna untuk menciptkan QUEUE yang baru dan kosong yaitu dengan cara memberikan nilai awal (Head) dengan satu (1) dan nilai akhir (Tail) dengan jumlah maksimal data yang akan ditampung/array. Berikut penggalan procedure Create:

```
Procedure Create;
Begin
      Head := 1; Tail := MaxQueue;
End;
```

2. Empty

Function Empty berguna untuk mengecek apakah QUEUE masih kosong atau sudah berisi data. Hal ini dilakukan dengan mengecek apakah Tail terletak bersebelahan dengan head, dan tail > head atau tidak, jika ya maka kosong. Berikut penggalan function empty:

```
Function Empty : Boolean;
Begin
      If ((Tail Mod MaxQueue) + 1 = Head) then
            Empty := True
      Else
            Empty := False;
End;
```

3. Full

Function Full berguna untuk mengecek apakah QUEUE sudah penuh atau masih bisa menampung data dengan cara mengecek apakah tempat yang masih kosong tinggal satau atau tidak (untuk membedakan dengan empty dimana semua tempat kosong), jika ya maka penuh. Berikut penggalan function full :

```
Function Full : Boolean;
Var
      x : 1..MaxQueue;
begin
      x := (Tail Mod MaxQueue) + 1;
      If ((x Mod MaxQueue) + 1) = Head) then
            Full := True
      Else
            Full := False;
End;
```

4. EnQueue

Procedure EnQueue berguna untuk memasukan 1 elemen kedalam QUEUE. (Tail dan Head mula-mula adalah nol(0)). Berikut penggalan procedure EnQueue:

```
Procedure EnQueue(Elemen : TypeElemen);
Begin
      If (Not Full) then
      Begin
            Tail := (Tail Mod MaxQueue) + 1;
            Queue[Tail] := elemen;
      End;
End;
```

5. DeQueue

Procedure DeQueue berguna untuk mengambil 1 elemen dari QUEUE. Hal ini dilakukan dengan cara memindahkan posisi head satu langkah ke belakang. Berikut penggalan procedure DeQueue:

```
Procedure DeQueue;
Begin
      If (Not Empty) then
            Head := (Head Mod MaxQueue) + 1;
End;
```

Implementasi QUEUE dengan DOUBLE LINKED LIST

Selain menggunakan array, Queue juga dibuat dengan linked list. Metode linked list yang digunakan adalah double linked list. Berikut penggalan type, konstanta dan variable yang akan digunakan dalam penjelasan operasi-operasi queue dengan linked list :

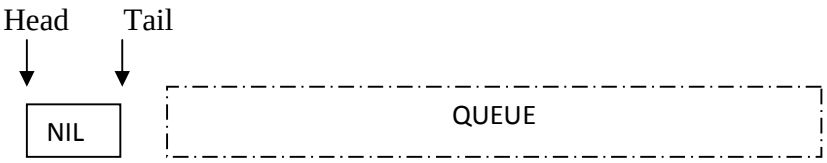
```
Type
      Point = ^Simpul;
      Simpul = Record
                  Isi : TipeData;
                  Next : Point;
               End;
      Queue = Record
                  Head : Point;
                  Tail : Point;
               End;
Var
      Q : Queue;
      N : 0..MaxQueue;  { Jumlah Antrian}
```

Operasi-operasi pembuatan QUEUE dengan DOUBLE LINKED LIST

1. Create

Procedure Create berguna untuk menciptakan QUEUE yang baru dan kosong yaitu dengan cara mengarahkan pointer head dan tail kepada NIL. Berikut penggalan procedure Create :

```
Procedure Create;
Begin
      Q.Head := NIL;      Q.Tail := Q.Head;
End;
```



2. Empty

Function Empty berguna untuk mengecek apakah QUEUE masih kosong atau sudah berisi data. Hal ini dilakukan dengan mengecek Head masih menunjuk pada nil atau tidak, jika ya maka kosong. Berikut penggalan function empty :

```
Function Empty : Boolean;
Begin
      If (Q.Head = NIL) then
            Empty := True
      Else
            Empty := False;
End;
```

3. Full

Function Full berguna untuk mengecek apakah QUEUE sudah penuh atau masih bisa menampung data. Hal ini dilakukan dengan mengecek apakah N (Jumlah Queue) sudah sama dengan Max_Queue atau belum, jika ya maka penuh. Berikut penggalan function full.

```
Function Full : Boolean;
Begin
      If (N = Max_Queue) then
            Full := True
      Else
            Full := False;
End;
```

4. EnQueue

Procedure EnQueue berguna untuk memasukan 1 elemen ke dalam QUEUE. (head dan tail mula-mula menunjuk ke NIL). Berikut penggalan procedure EnQueue.

```
Procedure EnQueue (Elemen : TipeData);
Var
      Now : Point;
Begin
      If (Not Full) then
      Begin
            New(Now);
            Now.^Isi := elemen;
            Now.^Next := NIL;
            If (Empty) then
            Begin
                  Q.Head := Now;
                  Q.Tail := Now;
```

```
                  N := 1;
            End else
            Begin
                  Q.Tail.^Next := Now;
                  Q.Tail := Now;
                  Inc(N);
            End;
      End;
End;
```

5. DeQueue

Procedure DeQueue beguna untuk mengambil 1 elemen dari QUEUE. Hal ini dilakukan dengan cara menghapus satu simpul yang terletak paling depan (depan). Berikut penggalan procedurenya.

```
Procedure DeQueue;
Var
      Now : Point;
Begin
      If (Not Empty) then
      Begin
            Now := Q.Head;
            Q.Head := Q.Head^.Next;
            Dispose(Now);
            Dec(N);
      End;
End;
```

Contoh Program antrian :

```
Uses Crt;
Const
      Max = 10;
Type
      Node  = ^Queue;
      Queue = Record
                  kar : char;
                  next : node;
              end;
Var
      pil   : char;
      Jml   : byte;
      Head, Now, Tail   : Node;
Procedure Push (ch : char);
Begin
      New(Now);
      If (head = nil) then
            Head := Now
      Else
            Tail^.next := Now;
      Tail := Now;
      Tail^.next := nil;
      Now^.kar := ch;
```

```
End;

Procedure Pop;
Begin
      Now := Head;
      Head := Head^.next;
      Dispose(Now);
End;

Procedure EnQueue;
Var
      i     : byte;
      temp  : char;
begin
      GotoXY(1,6); ClrEol; Write('Masukan @ karakter  : ');
      Repeat
            GotoXY(25,6); ClrEol;  temp := ReadKey; write(temp);
      Until (temp <> ' ');
      Push(temp);
      For (I := 1) to (75-Jml*6) do
      Begin
            GotoXY(i+1,20); write(' O');
            GotoXY(i,21); write('=(', Now^.kar, ')=');
            GotoXY(i+1,22); write('/  \'); delay(10);
            If (I <> 75-Jml*6) then
            Begin
                  GotoXY(i+1,20); write(' ');
                  GotoXY(i, 21); write('     ');
                  GotoXY(i+1,22); write(' ');
            End;
      End;
End;

Procedure DeQueue;
Var
      i, byk      : byte;
begin
      Now := Head;
      For (i :=  69) to 76 do
      Begin
            GotoXY(i+1,20); write(' O');
            GotoXY(i,21); write('=(', Now^.kar, ')=');
            GotoXY(i+1,22); write('/  \'); delay(10);
            GotoXY(i+1,20); write(' ');
            GotoXY(i, 21); write('     ');
            GotoXY(i+1,22); write(' ');
      End;
byk := 0;
while (byk <> Jml) do
begin
      inc (byk);
      Now := Now^.next;
      For (i := 69-byk*6) to (75-byk*6) do
      Begin
            GotoXY(i+1,20); write(' O');
            GotoXY(i,21); write('=(', Now^.kar, ')=');
            GotoXY(i+1,22); write('/  \'); delay(25);
```

```
            If (i <> 75-byk*6) then
            Begin
                  GotoXY(i+1,20); write(‘ ‘);
                  GotoXY(i, 21); write(‘     ‘);
                  GotoXY(i+1,22); write(‘ ‘);
            End;
      End;
End;
End;

Procedure Input;
Begin
      GotoXY(1, 1); Writeln(‘1. EnQueue ’);
      GotoXY(1, 2); Writeln(‘2. DeQueue ’);
      GotoXY(1, 3); Writeln(‘3. Exit ’);
      Repeat
            Repeat
                  GotoXY(1, 4); ClrEol; Write(‘Your Choice  : ‘);
                  Pil := ReadKey; Write(Pil);
            Unitl (Pil in [‘1’, ‘2’, ‘3’]);
            Case Pil of
            ‘1’   : Begin
                    If (Jml < Max) then
                        Begin
                              Inc (Jml);
                              EnQueue;
                        End else
                        Begin
                              GotoXY(1,   8);   Write(‘Antrian   penuh   !’);
delay(500);
                              GotoXY(1, 8); ClrEol;
                        End;
                  End;

            ‘2’   : Begin
                    If (Jml >= 1) then
                        Begin
                              Dec(Jml);
                              DeQueue;
                              Pop;
                        End else
                        Begin
                              GotoXY(1,   8);   write(‘Antrian   Kosong   !’);
delay(500);
                              GotoXY(1, 8); ClrEol;
                        End;
                        End;
                  End;
      Until (Pil = ‘3’);
End;

Begin
      Jml := 0;
      ClrScr;
      Input;
End.
```

7 GRAPH

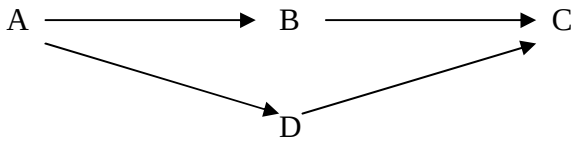
Graph merupakan himpunan tidak kosong dari node (vertek) dan garis penghubung (edge). Contoh sederhana tentang graph yaitu antara Fakultas Teknik dan mesjid UNSIL. Fakultas teknik dan mesjid merupakan node (vertek). Jalan yang menghubungkan Fakultas Teknik dan Mesjid merupakan garis penghubung antara keduanya (edge).

Fak.Teknik ——————————— Mesjid
Vertek *edge* *vertek*

Berdasarkan arahnya, graph dibagi menjadi dua :

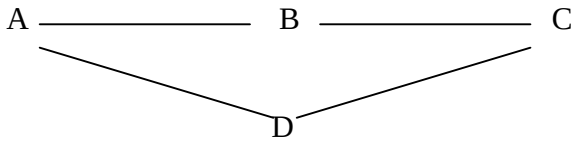
1) Gaph berarah (*directed graph*)

   Merupakan graph dimana tiap node memiliki edge yang memiliki arah, menunjukan kemana node tersebut dihubungkan.



2) Graph tak berarah (*undirected graph*)

   Merupakan graph dimana tiap node memiliki edge yang dihubungkan ke node lain tanpa arah.



   Selain arah, beban atau nilai sering ditambahkan pada edge. Misalnya nilai yang merepresentasikan panjang atau biaya transportasi dan lain-lain.

   Hal mendasar lain yang perlu diketahui adalah suatu node A dikatakan bertetangga dengan node B jika antara dengan B, B bertetangga dengan C, A tidak bertetangga dengan C, B tidak bertetangga dengan D.

   Masalah ketetanggaan ini harus benar-benar dikuasai implementasinya dalam program. Karena dengan ketetanggaan ini kita dapat mengetahui suatu node terhubung dengan node

yang mana, dengan demikian kita bisa melakukan proses-proses yang menggunakan data ketetanggaan ini, misalnya untuk mencari lintasan terpedek, pencarian dan lain-lain.

Untuk membangun graph kita dapat menggunakan arrat atau linked list.

Studi kasus GRAPH.

Misalkan kita akan menari daftar keterhubungan suatu kota dengan kota lainnya yang dihubungkan dengan suatu jembatan. Kota adalah node (vertek) dan jembatan adalah busur penghubung (edge).

Skenario yang dikehendaki adalah sebagai berikut :

1. User menginputkan data kota, elemen data kota disesuaikan dengan kebutuhan
2. User menginput data keterhubungan, yaitu dengan memasukan
   a. Kota Asal
   b. Kota Tujuan
   c. Jembatan yang menghubungkan
   d. Panjang jembatan
3. Program mengoutputkan data kota dan data keterhubungan, misalnya kita memiliki data kota Kota A, Kota B, Kota C dengan matrik ketetanggaan beserta jaraknya adalah sebagai berikut:

Node	A	B	C
A	~	5,3 Mil	1,2 Mil
B	5,3 Mil	~	4,2 Mil
C	1,2 Mil	4,2 Mil	~

Maka output program adalah diharapkan sebagai berikut:

Kota A, dengan data xyz

Ke Kota B 5,3 Mil

Ke Kota C 1,2 Mil

Kota B, dengan data uvw

Ke Kota A 5,3 Mil

Ke Kota C 4,2 Mil

Kota C, dengan data stu

Ke Kota A 1,2 Mil

Ke Kota B 4,2 Mil

Listing program untuk studi kasus diatas :

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

#define MaxCity 10
Char Response;
Struct Bridge {
   Char BridgeName[80];
   Float Mileage;
   Int Active;
};

Struct CityStruct{
   Char CityName[20];
   Float Population;
};

Struct CityStruct City[MaxCity];
Int CityCount;
Struct Bridge Marix[MaxCity][Maxcity];

Int GetCityIndex (char CityName[20], int*FoundFlag);

Void main() {
   Clrscr();
   Int i, j ; float P;
   Int FoundFlag, ToCityIndex, FromCityIndex;
   Char NameVar[80];
   Char FromName[80];
   Char ToName[80];
   Char BridgeName[80];
   Float Mileage;

   For (i=0; I < MaxCity; i++) {
         For (j=0; j<MaxCity; j++)
               Matrix[i][j].Active=0; }

Printf (“Input Data Kota(y/t) ?”); Response=getche();
CityCount=0;
While (Response==’Y’ || Response==’y’) {
   Printf(“\nMasukan Nama Kota   : “); gets(NameVar);
   Strcpy(City[CityCount].CityName,NameVar);
   Printf(“Masukan populasi penduduk   : “); scanf(“%f”,&P);
   City[CityCount].Populasi=P;
   Ffush(stdin);

   CityCount++;
   Printf(“\nLanjut input data Kota(y/t) ? “); Resposnse=getche();
   Printf(“\n===================================”);
   Printf(“\n\nInput Data keterhubungan(y/t) ?”); Response=getche();
   While (Response==’Y’ || Response==’y’) {
         Printf(“\nMasukkan Nama Kota Asal   : “); gets(FromName);
         Printf(“\nMasukkan Nama Kota Tujuan :”) gets(ToName);

         Printf(“\nMasukkan Nama Jembatan    : “); gets(BridgeName);
```

```
            Printf(“\nMasukkan Panjang Jembatan : ”); scanf(“%f”,&Mileage);

            From CityIndex = GetCityIndex(FromName, &FoundFlag);

            If (FoundFlag)    {
                  ToCityIndex = GetCityIndex (ToName, &FoundFlag);
                  If (FoundFlag) {
                        Marix[FromCityIndex][ToCiyIndex].Active=1;
                        Strcpy(Matrix[FromCityIndex][ToCityIndex].BridgeName,
   BridgeName);
                        Matrix[FromCityIndex][ToCityIndex].Mileage=Mileage;

                        Marix[toCityIndex][FromCityIndex].Active=1;

      Strcpy(Matrix[ToCityIndex][FromCityIndex].BridgeName,BridgeName);
                        Matrix[ToCityIndex][FromCityIndex].Mileage=Mileage;
                  }
            Else
            Printf(“City %s is not found!!!\n”, ToName);
      }
      Else
      Printf(“City %s is not found!!!\n”, FromName);
   Printf(“\nLanjut Data keterhubungan(y/t)? “); Response=getche();
   Fflush(stdin);
   }

   Printf(“\n=============================================”);
   Printf(“\nREPORT :\n---------------------------------\n”);
   For (i=0; i<CityCount; j++) {
      Printf(“Kota      %s      dengan      Populasi      =%.0f      orang”,
   City[i].CityName,City[i].Population);
      For (j=0; j<CityCount; j++) {
            If (Matrix[i][j].Active==1)
                  Printf(“\n\tKe  kota  %s  %.0f  Miles”,  City[j].CityName,
   Matrix[i][j].Mileage);
      }
      Printf(“\n”);
   }
   Getch();
}
Int GetCityIndex(char CityName[20], int*FoundFlag) {
      Int i=0;
      Int ReturnThis;

      *FoundFlag=0;
      While (I < CityCount) {
            If (strcmp(CityName, City[i].CityName==0) {
                  *FoundFlag=1;
                  ReturnThis=I; }
            i++;
      }
Return ReturnThis;
}
```

8 TREE

Tree merupakan salah satu bentuk struktur data tidak linier yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lain (disebut Subtree). Untuk jelasnya dibawah ini akan diuraikan istilah-istilah umum dalam tree.

a) Predecessor: node yang berada di atas node tertentu

b) Successor : node yang berada dibawah node tertentu

c) Ancestor : seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama

d) Descendent : seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama

e) Parent : predecessor satu level diatas suatu node

f) Child : successor satu level dibawah suatu node

g) Sibling : node-node yang memiliki parent yang sama dengan suatu node

h) Subtree : bagian dari tree yang berupa suatu node beserta descendannya dan memiliki semua karakteristik dari tree tersebut

i) Size : banyaknya node dalam suatu tree

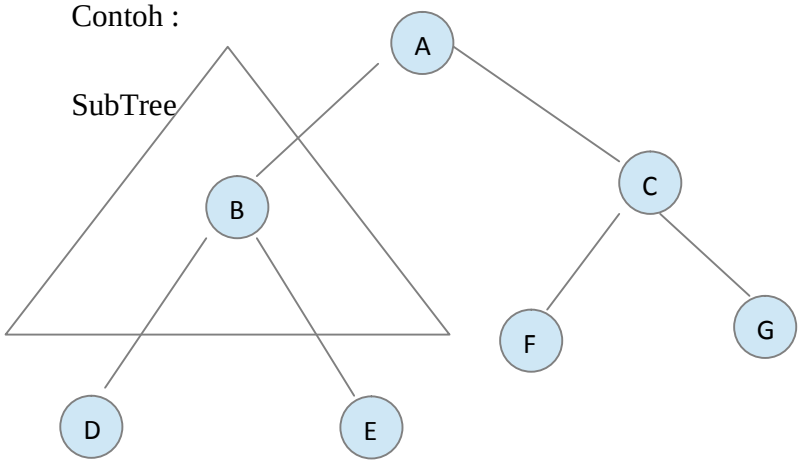
j) Height : banyaknya tingkatan/level dalam suatu tree

k) Root : satu-satunya node khusus dalam tree yang tak punya predecessor

l) Leaf : node-node dalam tree yang tak memiliki successor

m) Degree : banyaknya child yang dimiliki suatu node.

Contoh :



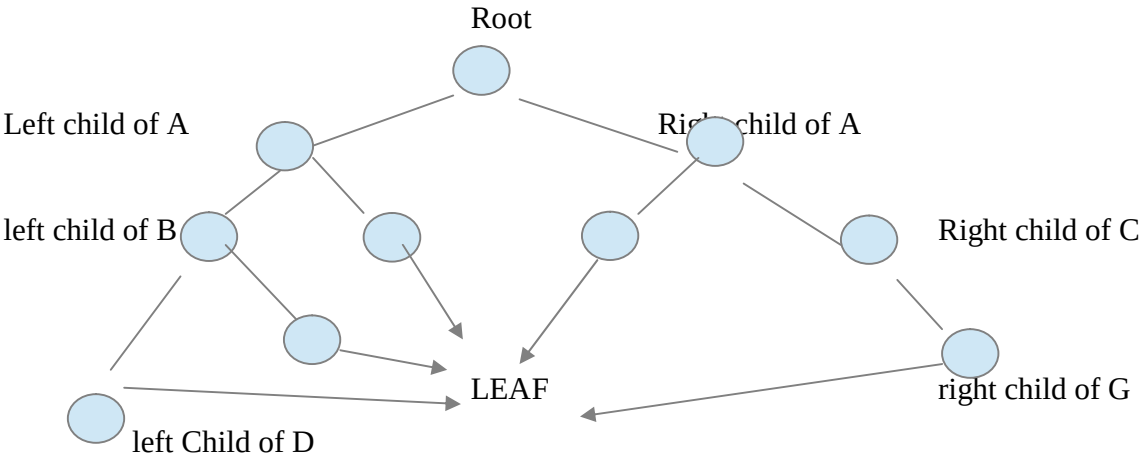
SubTree

Ancestor (F) = C, A
Descendent (C)= F, G
Parent (D) = B
Child (A) = B, C
Sibling (F) = G
Size = 7
Height = 3
Root = A
Leaf = D, E, F, G
Degree (C) = 2

Berberapa jenis Tree yang memiliki sifat khusus :

1. Binary Tree

Binary tree adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Sesuai dengan definisi tersebut, maka tiap node dalam binary tree hanya boleh memiliki paling banyak dua child.



Jenis-jenis binary tree :

a Full Binary Tree

Binary Tree yang tiap nodenya (kecuali leaf) memiliki dua child dan tiap subtree harus mempunyai panjang path yang sama.

b Complete Binary Tree

Mirip dengan Full binary tree, namun tiap subtree boleh memiliki panjang path yang berbeda. Node kecuali leaf memiliki 0 atau 2 child.

c Skewed Binary Tree

Yakni binary tree yang semua nodenya (kecuali leaf0 hanya memiliki satu child.

Operasi-operasi pada Binary Tree :

1. **Create :** membentuk binary tree baru yang masih kosong
2. **Clear :** mengosongkan binary tree yang sudah ada
3. **Empty :** function untuk memeriksa apakah binary tree masih kosong

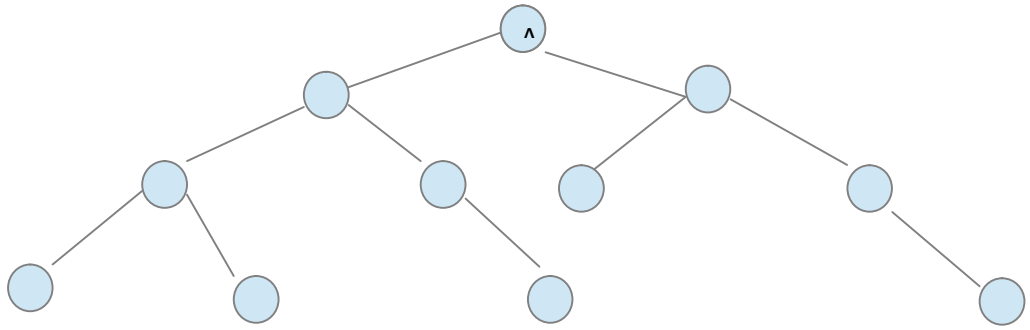
4. **Insert :** memasukan sebuah node kedalam tree. Ada tiga pilihan insert :sebagai root, left child, right child. Khusus insert sebagai root, tree harus dalam keadaan kosong.
5. **Find :** mencari root, parent, left child, right child dari suatu node.
6. **Update :** mengubah isi dari node yang ditunjukoleh pointer current.
7. **Retrieve :** mengetahui isi dari node yang ditunjuk oleh pointer current.
8. **DeleteSub :** menghapus sebuah subtree (node beserta seluruh descendantnya) yang ditunjuk current. Tree tak boleh kosong. Setelah itu pointer current akan berpindah ke parent dari node yang dihapus.
9. **Characteristic :** mengetahui karakteristik dari suatu tree,yakni : size, heigth, serta average lengthnya.
10. **Traverse :** mengunjungi seluruh node-node pada tree masing-masing sekali. Hasilnya adalah urutan informasi secara linier yang tersimpan dalam tree. Ada tiga cara traverse : Pre Order, In Order dan Post Order.

Langkah-langkah traverse :

- PreOrder : cetak isi node yang dikunjungi, kunjungi left Child, kunjungi Right Child
- InOrder : kunjungi left Child, cetak isi node yang dikunjungi, kunjungiRight Child
- PostOrder : kunjungi left Child, kunjungi Right Child, cetak isi node yang dikunjungi

2. Binary Search Tree

Adalah binary tree dengan sifat bahwa semua left child harus lebih kecil daripada right child dan parentnya. Juga semua right child harus lebih besar dari left child serta parentnya. Binary search tree dibuat untuk mengatasi kelemahan pada binary tree biasa, yaitu kesulitan dalam searching/pencarian node tertentu dalam binary tree. Contoh binary search tree umum:



Pada dasarnya operasi dalam binary serach tree sama dengan binary tree biasa, kecuali pada operasi insert, update dan delete.

- Insert : Pada binary search tree, insert dilakukan setelah ditemukan lokasi yang tepat.
- Update : Seperti pada binary tree biasa, namun di sini update akan berpengaruh pada posisi node tersebut selanjutnya. Bila setelah update mengakibatkan tree tersebut bukan binary search tree lagi, maka harus dilakukan perubahan pada tree dengan melakukan rotasi supaya tetap menjadi binary search tree.
- Delete : seperti halnya update, delete dalam binary search tree juga turut memepengaruhi struktur dari tree tersebut.

Contoh Program :

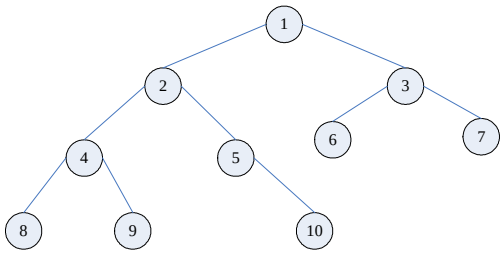
```
    program Binary_Tree;
    uses crt;
    type
      Ptr : ^Pohon;
      Pohon  : Record
                   isi   : byte;
             Left,Right : Ptr;
               end;
    var
         Root,Now : Ptr;
         x,y,cari : byte;
    procedure Push (var Tree : Ptr; data,Level : byte);
    var
      Now : Ptr;
    begin
      if (Tree = nil) then
      begin
            new(Now);
            Now^.isi := data;
            Now^.Left := nil;
            Now^.Right := nil;
            Tree := Now;
    end else
    if (data < Tree^.isi) then
      Push (Tree^.Left, data, Level)
    else
      if (data >= Tree^.isi) then
          Push (Tree^.Right, data, Level);
end;

procedure Input (var Tree : Ptr; Nilai, Selisih, Level : byte);
var
    x  : byte;
begin
    if (Level < 6) then
    begin
       Push (Tree, Nilai, Level);
       Input(Tree^.Left, Nilai-Selisih, Selisih div 2, Level+1);
```

```
      Input(Tree^.Rigth, Nilai+Selisih, Selisih div 2, Level+1);
    end;
end;

procedure Show (var Tree:Ptr; x,y,sel: byte);
var
    i : byte;
begin
    GotoXY(x,y); write(Tree^.isi);
    if (Tree^.Left <> nil) or (Tree^.Right <> nil)then
    begin
      GotoXY(x-sel, y+1); write('|^');
      GotoXY(x+sel, y+1);write('^|')';
      for (i := (x-sel)+1) to (x+sel)-1 do
      begin
            GotoXY(i, y+1); write('-');
        end;

GotoXY(x,y+1);write('_|_');
    end;
    inc (y,2);
    if (Tree^.Left <> nil) then
      show (Tree^.left, x-sel, y, sel div 2);
    if (Tree^.Right, x+sel, y, sel div 2);
end;

procedure Search (var Tree :Ptr; X,Y, Selisih, Cari : byte);
begin
    inc(y,2);
    if (cari < Tree^.isi) then
    begin
      if (Tree^.Left <> nil) then
            Search(Tree^.Left, X-Selisih, Y, Selisih div 2, Cari)
    end else
    if (Cari > Tree^.isi) then
    begin
      if (Tree^.Right<> nil) then
            Search(Tree^.Right, X-Selisih, Y, Selisih div 2, Cari)
    end else
    if (Cari = Tree^.isi) then
    begin
      dec(Y,2);
      GotoXY(X,Y); textcolor(10); write(Cari); ReadKey;
      GotoXY(X,Y); textcolor(15); write(Cari);
    end;

if ((Tree^.Left = nil) or (Tree^.Right = nil)) and (Cari <> Tree^.isi) then
begin
    GotoXY(2,2); write('Data Tidak Ada'); ReadKey;
    GotoXY(2,2); ClrEol;
end;

procedure CursorOff; assembler;
asm
    mov ah,1
    mov cx,0100h;
    int 10h;
```

```
end;

procedure CursorOn; assembler;
asm
    mov ah,1
    mov cx,0607h
    int 10h;
end;
begin
    ClrScr;
    TextColor(15);
    Randomize;
    x := random(20);
    Input (Root, x+30, (x+30) div 2, 1);
    repeat
       Show (Root, 40, 5, 20);
       GotoXY(2,1);ClrEol; Write('Find (0 = Quit) : '); Readln(Cari);
       Now := Root; x:= 40; y := 5;
       CursorOff;
       if (Cari <> 0) then
             Search(Now,X,Y,20,Cari);
             CursorOn;
    unitl (Cari=0);
end.
```

Contoh Soal Latihan :

1. Jelaskan dan sebutkan perbedaan anatar Struktur Data List, Stack, Queue, Graph dan Tree!
2. Sebutkan operasi-operasi apa saja yang bisa dilakukan dalam Queue. Dan jelaskan pula fungsi dari tiap-tiap operasinya.
3. Tuliskan dua metode yang bisa digunakan untuk mencari jalur terpendek dari Graph. Jelaskan!
4. Tuliskan contoh procedure untuk menambahkan elemen kedalam stack.
5. Tuliskan dan jelaskan langkah-langkah traverse yang bisa digunakan dalam Tree!
6. Penelusuran pohon binner :



Tuliskan algoritma untuk penelusuran pohon binner menggunakan PreOrder untuk mencapai node 7.