Queue (Antrian)

Tim Pengajar IF2030

Queue

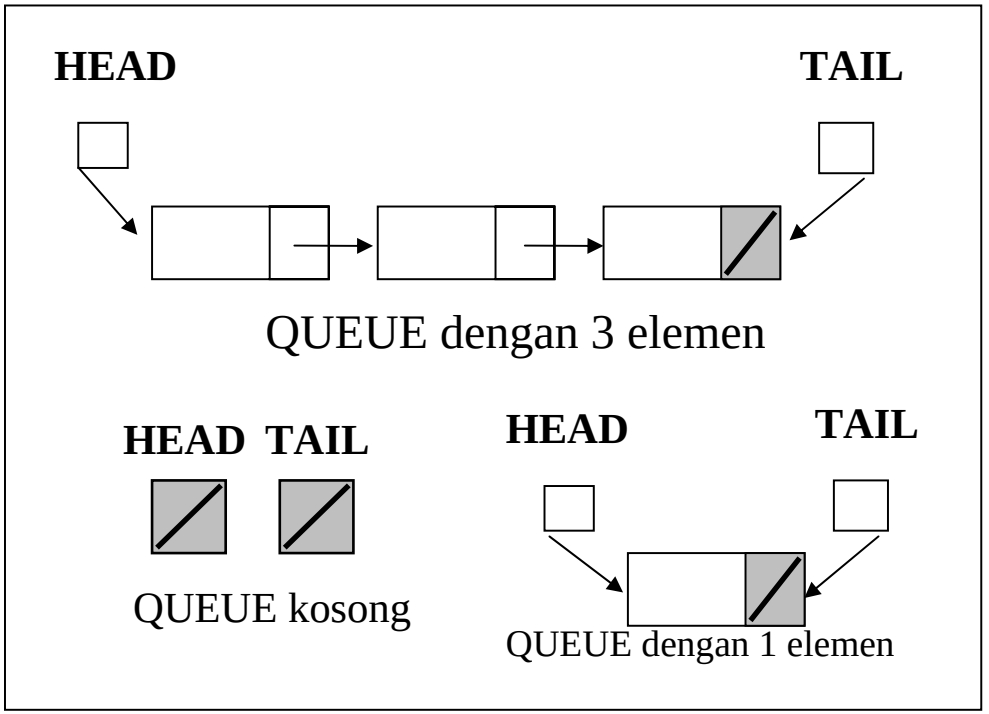
- QUEUE adalah list linier yang:
  - dikenali elemen pertama (HEAD) dan elemen terakhirnya (TAIL)
  - aturan penyisipan dan penghapusan elemennya didefinisikan sebagai berikut:
    - Penyisipan selalu dilakukan setelah elemen terakhir
    - Penghapusan selalu dilakukan pada elemen pertama
  - satu elemen dengan yang lain dapat diakses melalui informasi NEXT

Queue

- Elemen Queue tersusun secara FIFO *(First In First Out)*
- Pemakaian queue:
  - antrian job yang harus ditangani oleh sistem operasi (job scheduling)
  - antrian dalam dunia nyata

Queue

- Secara lojik:
  - Head
  - Tail
  - Elemen
  - Queue kosong



Definisi Fungsional

- Jika diberikan **Q** adalah QUEUE dengan elemen **ElmtQ**

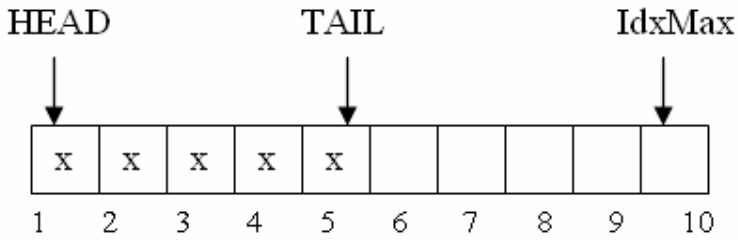
```
IsEmpty        : Q → boolean   { Tes terhadap Q: true jika Q kosong,
                                 false jika Q tidak kosong }
IsFull         : Q → boolean   { Tes terhadap Q: true jika memori Q
                                 sudah  penuh,
                                 false jika memori Q tidak penuh }
NBElmt(Q)      : Q → integer   { Mengirimkan banyaknya elemen Q }
CreateEmpty    : → Q           { Membuat sebuah antrian kosong }
Add            : ElmtQ x Q → Q { Menambahkankan sebuah elemen setelah
                                 elemen ekor QUEUE }
Del            : Q → Q x ElmtQ { Menghapus kepala QUEUE,
                                 mungkin Q menjadi kosong }
```

Implementasi Queue dengan Tabel

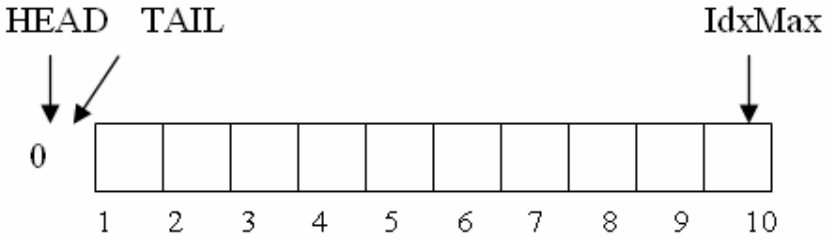
- Memori tempat penyimpan elemen adalah sebuah tabel dengan indeks 1..IdxMax.
- IdxMax dapat juga “dipetakan” ke kapasitas Queue
- Representasi field Next: Jika i adalah “address” sebuah elemen, maka suksesor i adalah Next dari elemen Queue.

Implementasi Queue dengan Tabel Alternatif I

- Jika Queue tidak kosong: TAIL adalah indeks elemen terakhir, HEAD selalu diset = 1
- Jika Queue kosong, maka HEAD = 0
- Ilustrasi Queue tidak kosong dengan 5 elemen



- Ilustrasi Queue kosong:

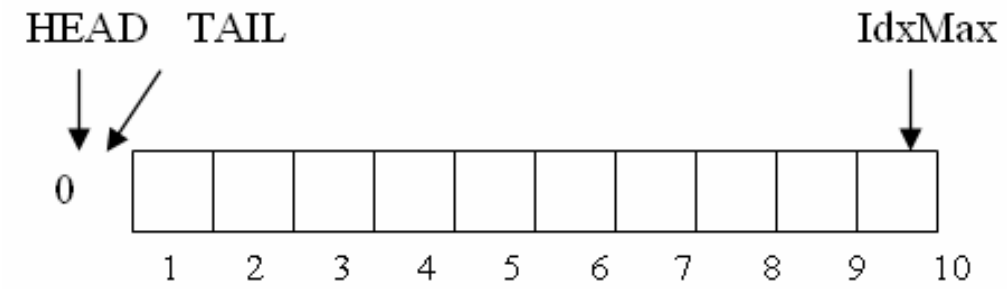


Implementasi Queue dengan Tabel Alternatif I

- Algoritma **penambahan elemen**:
  - Jika masih ada tempat, “majukan” TAIL
  - Kasus khusus untuk Queue kosong: karena HEAD harus diset nilainya menjadi 1
- Algoritma paling sederhana dan “naif” untuk **penghapusan elemen**
  - Jika Queue tidak kosong: ambil nilai elemen HEAD, geser semua elemen mulai dari HEAD+1 s.d. TAIL (jika ada), kemudian TAIL “mundur”
  - Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI
- Algoritma ini mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi tidak efisien

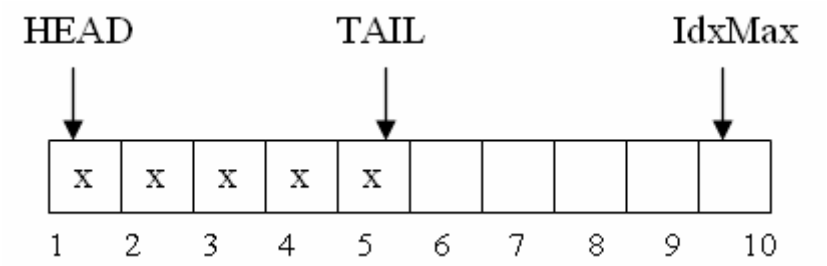
Implementasi Queue dengan Tabel Alternatif II

- Tabel dengan representasi HEAD dan TAIL, HEAD “bergerak” ketika sebuah elemen dihapus
- Jika Queue kosong, maka HEAD = 0
- Ilustrasi Queue kosong:

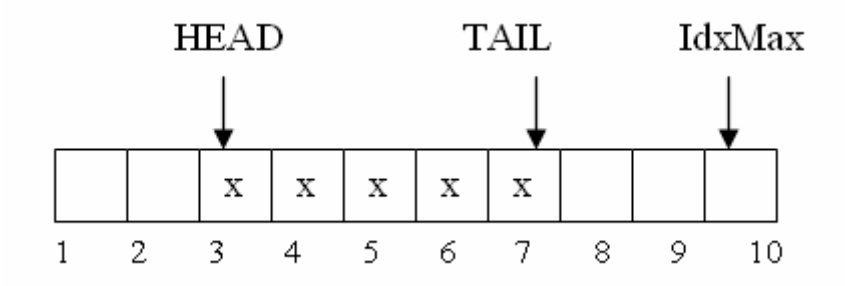


Implementasi Queue dengan Tabel Alternatif II

- Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan pertama HEAD “sedang berada di posisi awal:



- Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan pertama HEAD tidak berada di posisi awal (akibat algoritma penghapusan)

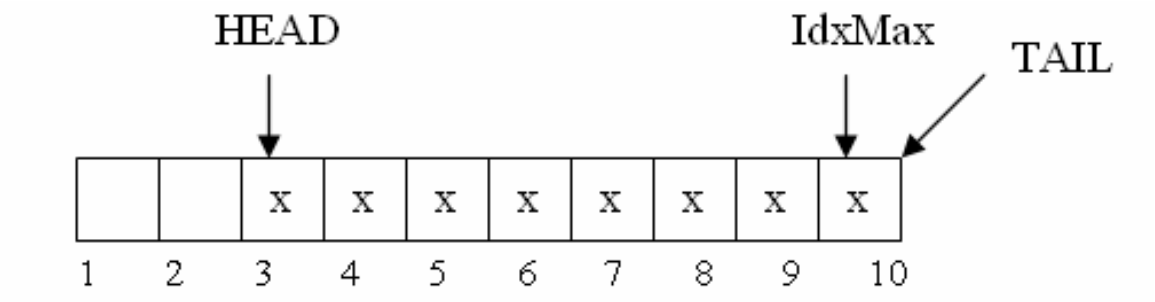


Implementasi Queue dengan Tabel Alternatif II

- Algoritma **penambahan elemen** sama dengan alternatif I
- Algoritma **penghapusan elemen**:
  - Jika Queue tidak kosong: ambil nilai elemen HEAD, kemudian HEAD "maju"
  - Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI.
- Algoritma ini **TIDAK** mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi **efisien**

Implementasi Queue dengan Tabel Alternatif II

- Keadaan Queue penuh tetapi “semu” sebagai berikut:



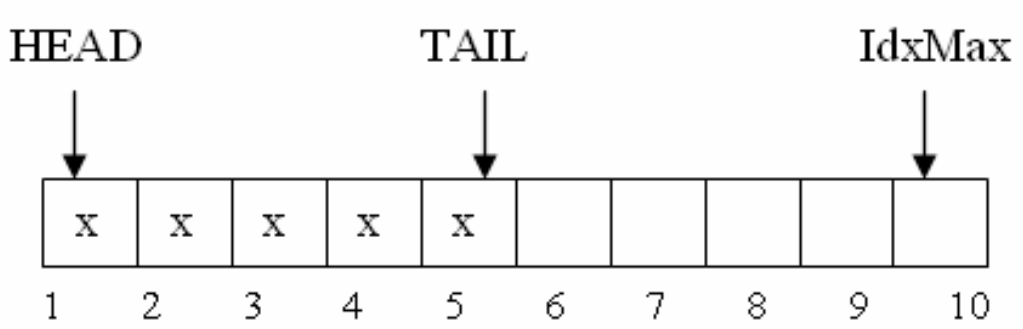
- Harus dilakukan aksi menggeser elemen untuk menciptakan ruangan kosong
- Pergeseran hanya dilakukan jika dan hanya jika TAIL sudah mencapai IdxMax

Implementasi Queue dengan Tabel Alternatif III

- Tabel dengan representasi HEAD dan TAIL yang “berputar” mengelilingi indeks tabel dari awal sampai akhir, kemudian kembali ke awal
- Jika Queue kosong, maka HEAD=0
- Representasi ini memungkinkan tidak perlu lagi ada pergeseran yang harus dilakukan seperti pada alternatif II pada saat penambahan elemen

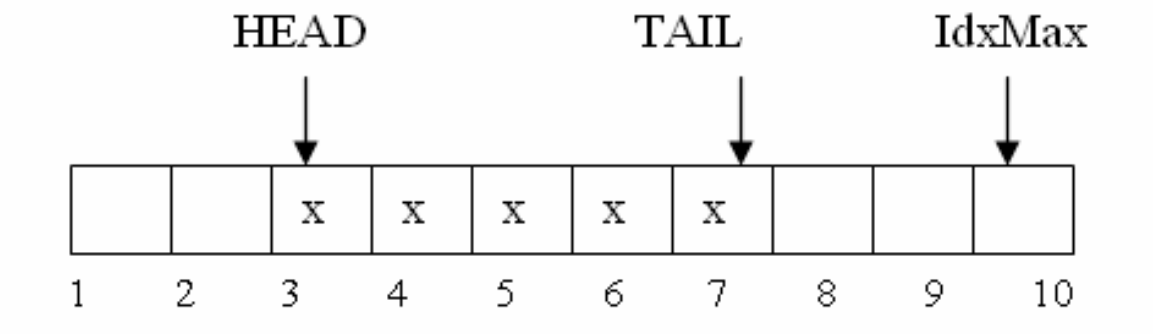
Implementasi Queue dengan Tabel Alternatif III

- Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD “sedang” berada di posisi awal



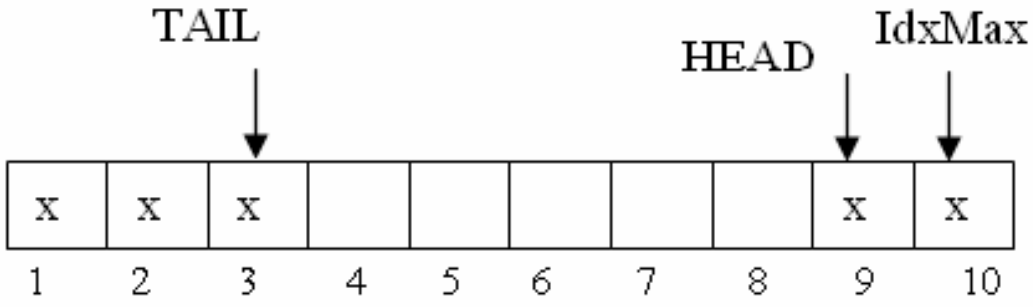
Implementasi Queue dengan Tabel Alternatif III

- Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD tidak berada di posisi awal, tetapi masih “lebih kecil” atau “sebelum” TAIL (akibat penghapusan/ penambahan)



Implementasi Queue dengan Tabel Alternatif III

- Ilustrasi Queue tidak kosong, dengan 5 elemen, HEAD tidak berada di posisi awal, tetapi “lebih besar” atau “sesudah” TAIL (akibat penghapusan/penambahan)



Implementasi Queue dengan Tabel Alternatif III

- Algoritma **penambahan elemen**:
  - Jika masih ada tempat, “majukan” TAIL
  - Jika TAIL sudah mencapai IdxMax, maka suksesor dari IdxMax adalah 1 sehingga TAIL yang baru adalah 1
  - Jika TAIL belum mencapai IdxMax, maka algoritma penambahan elemen sama dengan alternatif II
  - Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1

Implementasi Queue dengan Tabel Alternatif III

- Algoritma **penghapusan elemen**
  - Jika Queue tidak kosong: ambil nilai elemen HEAD, kemudian HEAD “maju”. Penentuan suatu suksesor dari indeks yang diubah/”maju” dibuat seperti pada algoritma penambahan elemen: jika HEAD mencapai IdxMAx, maka suksesor dari HEAD adalah 1
  - Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI.

Implementasi Queue dengan Tabel Alternatif III

- Algoritma ini **efisien** karena tidak perlu pergeseran, dan seringkali strategi pemakaian tabel semacam ini disebut sebagai “circular buffer”, di mana tabel penyimpan elemen dianggap sebagai “buffer”
- Salah satu variasi dari representasi pada alternatif III:
  - menggantikan representasi TAIL dengan COUNT (banyaknya elemen Queue)

ADT Queue dengan C - Alternatif II
Alokasi Memori Dinamik

```
/* File : queue.h */
#ifndef queue_H
#define queue_H
#include "boolean.h"
#include <stdlib.h>
#define Nil 0
/* Definisi elemen dan address */
typedef int infotype;
typedef int address;   /* indeks tabel */
/* Contoh deklarasi variabel bertype Queue : */
/* Versi I : tabel dinamik,Head dan Tail eksplisit, ukuran disimpan */
typedef struct { infotype *T;   /* tabel penyimpan elemen */
                 address HEAD;  /* alamat penghapusan */
                 address TAIL;  /* alamat penambahan */
                 int MaxEl;     /* MAx elemen queue */
               } Queue;
/* Definisi Queue kosong: Head=Nil; TAIL=Nil. */
/* Catatan implementasi: T[0] tidak pernah dipakai */
```

ADT Queue dengan C Alternatif II


```
/********** AKSES (Selektor) **********/
/* Jika Q adalah Queue, maka akses elemen : */
#define Head(Q) (Q).HEAD
#define Tail(Q) (Q).TAIL
#define InfoHead(Q) (Q).T[(Q).HEAD]
#define InfoTail(Q) (Q).T[(Q).TAIL]
#define MaxEl(Q) (Q).MaxEl
/********** Prototype **********/
boolean IsEmpty (Queue Q);
/* Mengirim true jika Q kosong: lihat definisi di atas */
boolean IsFull(Queue Q);
/* Mengirim true jika tabel penampung elemen Q sudah penuh */
/* yaitu mengandung MaxEl elemen */
int NBElmt(Queue Q);
/* Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong */
```

ADT Queue dengan C Alternatif II


```
/*** Kreator ***/
void CreateEmpty(Queue *Q, int Max);
/* I.S. sembarang */
/* F.S. Sebuah Q kosong terbentuk dan salah satu kondisi sbb: */
/* Jika alokasi berhasil, Tabel memori dialokasi berukuran Max */
/* atau : jika alokasi gagal, Q kosong dg Maksimum elemen=0 */
/* Proses : Melakukan alokasi,Membuat sebuah Q kosong */
/*** Destruktor ***/
void DeAlokasi(Queue *Q);
/* Proses: Mengembalikan memori Q */
/* I.S. Q pernah dialokasi */
/* F.S. Q menjadi tidak terdefinisi lagi, MaxEl(Q) diset 0 */
/*** Primitif Add/Delete ***/
void Add (Queue * Q, infotype X);
/* Proses: Menambahkan X pada Q dengan aturan FIFO */
/* I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh */
/* F.S. X menjadi TAIL yang baru, TAIL "maju" */
/* Jika Tail(Q)=MaxEl+1 maka geser isi tabel, shg Head(Q)=1 */
void Del(Queue * Q, infotype* X);
/* Proses: Menghapus X pada Q dengan aturan FIFO */
/* I.S. Q tidak mungkin kosong */
/* F.S. X = nilai elemen HEAD pd I.S.,HEAD "maju";Q mungkin kosong */
#endif
```

ADT Queue dengan C Alternatif II


```
boolean IsEmpty (Queue Q)
/* Mengirim true jika Q kosong: lihat definisi di atas */
{  /* Kamus Lokal */

   /* Algoritma */
   return ((Head(Q)==Nil) && (Tail(Q)==Nil));
}
boolean IsFull (Queue Q)
/* Mengirim true jika tabel penampung elemen Q sudah penuh */
/* yaitu mengandung MaxEl elemen */
{  /* Kamus Lokal */

   /* Algoritma */
   return ((Head(Q)==1) && (Tail(Q)==MaxEl(Q)));
}
int NBElmt(Queue Q)
/* Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong */
{  /* Kamus Lokal */

   /* Algoritma */
   return (Tail(Q)-Head(Q)+1);
}
```

ADT Queue dengan C Alternatif II


```
void CreateEmpty(Queue *Q, int Max)
/* I.S. sembarang */
/* F.S. Sebuah Q kosong terbentuk dan salah satu kondisi sbb: */
/* Jika alokasi berhasil, Tabel memori dialokasi berukuran Max */
/* atau : jika alokasi gagal, Q kosong dg Maksimum elemen=0 */
/* Proses : Melakukan alokasi, Membuat sebuah Q kosong */
/* yaitu mengandung MaxEl elemen */
{  /* Kamus Lokal */

   /* Algoritma */
   (*Q).T = (infotype *) malloc ((Max+1) * sizeof(infotype));
   if ((*Q).T != NULL) {
        MaxEl(*Q) = Max;
        Head(*Q) = Nil;
        Tail(*Q) = Nil;
   } else /* alokasi gagal */ {
        MaxEl(*Q) = Nil;
   }
}
```

ADT Queue dengan C Alternatif II


```
void DeAlokasi(Queue *Q)
/* Proses: Mengembalikan memori Q */
/* I.S. Q pernah dialokasi */
/* F.S. Q menjadi tidak terdefinisi lagi, MaxEl(Q) diset 0 */
{  /* Kamus Lokal */

   /* Algoritma */
   MaxEl(*Q) = Nil;
   free((*Q).T);
}
```

ADT Queue dengan C - Alternatif II

```
void Add (Queue *Q, infotype X)
/* Proses: Menambahkan X pada Q dengan aturan FIFO */
/* I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh */
/* F.S. X menjadi TAIL yang baru, TAIL "maju" */
/* Jika Tail(Q)=MaxEl+1 maka geser isi tabel, shg Head(Q)=1 */
/* yaitu mengandung MaxEl elemen */
{  /* Kamus Lokal */
   address i, j;
   /* Algoritma */
   if (IsEmpty(*Q)) {
      Head(*Q) = 1; Tail(*Q) = 1;
   } else /* Q tidak kosong */ {
      if (Tail(*Q)==MaxEl(*Q)) { /* Tail berada di MaxEl */
         i = Head(*Q); j = 1;
         do { *((*Q).T+j) = *((*Q).T+i);
              i++; j++;
         } while (i<=Tail(*Q));
         Head(*Q) = 1; Tail(*Q) = j-1;
      } else {
         Tail(*Q)++;
      }
   }
   InfoTail(*Q)=X;
}
```

ADT Queue dengan C Alternatif II


```
void Del(Queue * Q, infotype* X)
/* Proses: Menghapus X pada Q dengan aturan FIFO */
/* I.S. Q tidak mungkin kosong */
/* F.S. X = nilai elemen HEAD pd I.S.,HEAD "maju"; Q mungkin kosong */
/* yaitu mengandung MaxEl elemen */
{  /* Kamus Lokal */

   /* Algoritma */
   *X = InfoHead(*Q);
   if (Head(*Q)==Tail(*Q)) { /* Set mjd queue kosong */
      Head(*Q)=Nil; Tail(*Q)=Nil;
   }
   else {
      Head(*Q)++;
   }
}
```

ADT Queue dengan C
Alternatif III – Queue Sirkuler

- Diktat Struktur Data hlm. 63

PR

- Modul pra-praktikum bagian Pemrograman Prosedural:
  - P-09. ADT QUEUE
    - Bagian 1. Representasi Tabel Kontigu dengan Alokasi Memori Dinamik – Alternatif I
    - Bagian 2. Representasi Tabel Kontigu dengan Alokasi Memori Dinamik – Alternatif II
    - Bagian 3. Representasi Tabel Kontigu dengan Alokasi Memori Dinamik – Alternatif III