1.3 Pencarian Beruntun (*Sequential Search*)

Di dalam Buku 1 kita telah mempelajari algoritma pencarian yang paling sederhana, yaitu pencarian beruntun. Di dalam Bab ini kita kemukakan kembali pencarian beruntun karena skema pencarian beruntun nantinya akan diperluas untuk menurunkan algoritma pencarian dengan sentinel.

Pada dasarnya, pencarian beruntun adalah proses membandingkan setiap elemen larik satu per satu secara beruntun, mulai dari elemen pertama, sampai elemen yang dicari ditemukan, atau seluruh elemen sudah diperiksa. Terdapat dua macam pencarian beruntun; pertama, pencarian beruntun pada larik yang tidak terurut, dan kedua, pencarian beruntun pada larik terurut. Perbedaan kedua terletak pada kemangkusan (*efficiency*) operasi pembandingan yang dilakukan.

1.3.1 Pencarian Beruntun pada Larik yang Tidak Terurut

Pencarian dilakukan dengan memeriksa setiap elemen larik mulai dari elemen pertama sampai elemen yang dicari ditemukan atau sampai seluruh elemen sudah diperiksa.

Contoh: Perhatikan larik L di bawah ini:

13	16	14	21	76	21
1	2	3	4	5	6

```
Misalkan nilai  yang dicari adalah: X = 21
Maka, elemen yang diperiksa: 13, 16, 14, 21 (ditemukan!)
Indeks larik yang dikembalikan: IX = 4

Misalkan nilai  yang dicari adalah: X = 13
Maka, elemen yang diperiksa: 13 (ditemukan!)
Indeks larik yang dikembalikan: IX = 1

Misalkan nilai  yang dicari adalah: X = 15
Maka, elemen yang diperiksa: 13, 16, 14, 21, 76, 21 (tidak ditemukan!)
Indeks larik yang dikembalikan: IX = 0
```

Algoritma pencarian beruntun dituliskan di bawah ini:

```
{ kamus global }
KAMUS
   const Nmaks : integer = 100      {jumlah maksimum elemen larik }
   type Larik100 = array [1..Nmaks] of integer
```

```
procedure CariRuntun1(input L : Larik100, input N : integer,  input X:integer,
                      input/output IX : integer)
{ Mencari  X di dalam larik L[1..N] secara beruntun, dimulai dari elemen pertama
sampai X ditemukan atau seluruh elemen larik L sudah diperiksa. Keluaran prosedur
ini adalah indeks IX yang L[IX]= X. IX berharga 0 jika X tidak ditemukan.
K.Awal : Larik L[1..N] sudah terdefinisi harganya, X adalah harga yang akan
         dicari.
K.Akhir: IX berisi indeks larik tempat X ditemukan, IX = 0 jika X tidak
         ditemukan. }
```

```
KAMUS LOKAL
   I : integer      { indeks untuk pencarian}
```

```
ALGORITMA
   I ← 1

   { periksa selama I < N dan L[I] ≠ X }
   while (I ≤ N ) and (L[I] ≠ X) do
      I ← I + 1    { maju ke elemen berikutnya }
   endwhile
   { I = N or L[I] = X }

   { simpulkan hasil pencarian }
   if (L[I] ≠ X) then
     IX ← 0
   else
     IX ← I
   endif
```

Pada algoritma CariRuntun1 di atas, pembandingan X dengan elemen larik dilakukan di dalam kondisi kalang while-do. Apabila elemen larik yang ke-I tidak sama dengan X dan I belum sama dengan N, pemeriksaan diteruskan ke elemen berikutnya (I ← I + 1).
Pemeriksaan dihentikan apabila A[I] = X atau indeks I sudah sama dengan N. Hasil pencarian disimpulkan di luar kalang while-do dengan pernyataan if (A[I] ≠ X) then ... Pernyataan terakhir ini juga memeriksa apakah elemen terakhir, A[N], sama dengan X. Jadi, pada algoritma CariRuntun1 di atas, elemen terakhir diperiksa secara khusus.

Contoh: cara pemanggilan prosedur CariRuntun1:

```
{ Misalkan P dan IP sudah didefinisikan tipenya di dalam kamus }
input(P)
CariRuntun1(L,P,IP)
if IP = 0 then
  output(P,' tidak ditemukan!')
else
  { proses terhadal L[IP] }
  ...
endif
```

Versi kedua dari algoritma di atas adalah meletakkan perintah pembandingan X dengan elemen larik di dalam kalang while-do. Untuk itu, diperlukan peubah *boolean* yang menghentikan proses pemeriksaan apabila X sudah ditemukan. Misalkan peubah tersebut bernama ketemu yang pada mulanya diisi nilai false. Bila pada pembandingan A[I] = X, maka ketemu diisi dengan harga true. Proses pencarian dihentikan. Hasil pencarian disimpulkan di luar kalang while-do.