Algoritme dan Pemrograman

- Tipe struct
- *Searching* (Pencarian)
- *Bubble Sort* (Pengurutan)

Pengertian struct

- **struct** adalah kumpulan variabel (masing-masing dapat berbeda tipe) yang dikelompokkan dan dikemas ke dalam satu nama variabel.
- Untuk mendefinisikan suatu *record*.
- Termasuk tipe data turunan (*derived data type*).

Cara membuat struct

- Contoh:

```
struct dataMhs {
   char *nama;
   int usia;
};
```

- Tipe struct ini diberi nama dataMhs
- Terdiri atas dua variabel: nama dan usia
- Pernyataan tersebut **hanya** membuat tipe data struct baru, **TIDAK** mendeklarasikan variabel apapun
  - Belum ada alokasi memori

Deklarasi struct

- Contoh deklarasi variabel menggunakan tipe struct:

```
struct dataMhs mhs;
struct dataMhs arrMhs[100];
```

- Dapat juga dilakukan langsung (definisi dan deklarasi):

```
struct dataMhs {
   char *nama;
   int usia;
} mhs, arrMhs[100];
```


Contoh program :: Merekam data nama dan usia mhs.

```
#include <stdio.h>
struct dataMhs {
        char *nama;
        int usia;
} mhs1;

int main() {
   struct dataMhs mhs2={"Indah Sekali", 19};
   mhs1.nama = "Elok Nian";
   mhs1.usia = 20;
   printf("%s %d\n", mhs1.nama, mhs1.usia);
   printf("%s %d\n", mhs2.nama, mhs2.usia);
   return 0;
}
```

```
Elok Nian 20
Indah Sekali 19
```


Pointer pada struct

```
#include <stdio.h>
struct dataMhs {
        char *nama;
        int usia;
};

int main() {
   struct dataMhs mhs={"Indah Sekali", 19};
   struct dataMhs *p;
   p = &mhs;
   printf("%s %d\n", mhs.nama, mhs.usia);
   printf("%s %d\n", p->nama, p->usia);
   return 0;
}
```

```
Indah Sekali 19
Indah Sekali 19
```

Akses anggota struct

- Menggunakan salah satu dari dua operator:
  - operator titik (.)
  - operator panah (->)

  tergantung tipe variabel yang dideklarasikan.
- Akses variabel biasa (selain pointer) menggunakan operator titik, sedangkan akses variabel pointer menggunakan operator panah.

Typedef

- Typedef merupakan mekanisme untuk membuat sinonim atau alias dari tipe data yang telah didefinisikan sebelumnya.
- Contoh:

```
typedef struct dataMhs MHS;
```

  berarti mendefinisikan tipe data baru bernama MHS sebagai sinonim untuk "struct dataMhs".
- Dengan demikian, pernyataan struct dataMhs untuk selanjutnya dapat diganti dengan MHS saja.

Contoh :: Menggunakan typedef

```
#include <stdio.h>
struct dataMhs {
        char *nama;
        int usia;
};
typedef struct dataMhs MHS;

int main() {
   MHS mhs={"Indah Sekali", 19};
   MHS *p;
   p = &mhs;
   printf("%s %d\n", mhs.nama, mhs.usia);
   printf("%s %d\n", p->nama, p->usia);
   return 0;
}
```

```
Indah Sekali 19
Indah Sekali 19
```


Array of struct

```
#include <stdio.h>
#define SIZE 100
struct nilaiMhs {
   char nim[9];
   int uts, uas;
   float rataan;
};
typedef struct nilaiMhs NILAI;

void substring
(char *dest, const char *source, int a, int n) {
   int i=a;
   for (; i<a+n; i++)
      dest[i-a]=source[i];
   dest[i-a]='\0';
}
```


Array of struct

```
int main() {
   int n, i;
   NILAI list[SIZE];
   char brs[256], st[4];
   scanf("%d\n", &n);
   for (i = 0; i<n; i++) {
      gets(brs);  // baca per baris data
      substring(list[i].nim, brs, 0, 9);
      substring(st, brs, 10, 2); list[i].uts = atoi(st);
      substring(st, brs, 13, 2); list[i].uas = atoi(st);
   }

   for (i = 0; i<n; i++)   // hitung rataan tiap mhs
      list[i].rataan = (float) (list[i].uts+list[i].uas)/2.0;

   for (i = 0; i<n; i++) {
      printf("%s %.2f\n", list[i].nim, list[i].rataan);
   }
   return 0;
}
```

```
3
G64010234 60 80
G64010235 50 45
G64010236 90 76
```


Latihan #1 Titik 2D

- Suatu titik dalam bidang 2D diwakili oleh dua nilai yang masing-masingnya menunjukkan nilai untuk masing-masing dimensi dan dicetak dengan notasi (x, y) dengan x dan y adalah bilangan riil
- Buat program untuk membaca dua buah titik dan menampilkan jarak Euclid keduanya dengan dua angka di belakang titik desimal. Gunakan tipe data struct
- Contoh masukan:

```
2.5 2
5 2
```

$$d(\mathbf{p},\mathbf{q}) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

- Contoh keluaran:

```
2.50
```

Latihan #2
Waktu

- Buat fungsi untuk menentukan selisih antara dua waktu time1 dan time2 dalam **satuan menit**. Masing-masing waktu terdiri atas tiga komponen nilai, yaitu *hour* (jam), *minute* (menit), dan *second* (detik).
- Contoh penggunaan fungsi:

```
int main() {
   TIME time1 = {10, 30, 0.0};
   TIME time2 = {11, 31, 30.0};
   float beda;
   beda = selisih(time1, time2);
   printf(".2f\n", beda);   // output: 61.50
   return 0;
}
```


Latihan #3

- Diketahui suatu struct mhs untuk data mahasiswa:

```
struct mhs {
    char nim[10];
    char nama[30];
    float nilai;
};
```

- Buat suatu program untuk menerima n buah data mahasiswa dan menampilkan daftar mahasiswa yang terurut berdasarkan nilai secara *descending*

Data untuk *searching* & *sorting*

- Anggaplah kita mempunya *array* data yang terdiri dari beberapa kolom, salah satunya adalah key dengan tipe int sbb:

```
struct list {
   int key;
   int val;  // contoh elemen lainnya
   ... // dan seterusnya
};
typedef struct list LIST;
```

- Akan dilakukan sorting berdasarkan nilai *key* secara ascending.

Searching (pencarian)

- Mencari data berdasarkan nilai tertentu, x.
- Beberapa contoh algoritme pencarian:
  - *Sequential search*
  - *Sorted array search*
  - *Hashing*
  - *Recursive structures search*
  - *Multidimensional search*
- Yang dibahas pada mata kuliah ini adalah *sequential search* dan *sorted array search*
- Pencarian penting dalam struktur data

Variasi masalah

- Ada atau tidak ada? Keluaran: TRUE/FALSE
- Indeks kemunculan pertama? Keluaran: suatu nilai antara 0 dan n-1 jika ada atau suatu nilai khusus (mis. -1) jika tidak ada
- Frekuensi kemunculan x? Keluaran: antara 0 hingga n
- Indeks seluruh kemunculan jika ada
- Indeks kemunculan terakhir atau ke-i
- Dapat divariasikan juga dengan
  - Pencarian nilai yang lebih kecil atau lebih besar dari x
  - Pencarian nilai tertentu untuk fungsi dari x (mis. nilai yang selisihnya dengan x kurang dari suatu nilai ambang)
  - Penggunaan dua nilai x dan y untuk mencari data yang nilainya di antara keduanya

Sequential search

- Setiap elemen dalam *array* dibandingkan satu per satu dari awal (kiri) hingga x ditemukan atau semua elemen telah diperiksa.
- Mudah diimplementasikan.
- Waktu proses relatif lambat untuk ukuran data sangat besar.
  - Jika suatu *array* memiliki n buah elemen, sebanyak-banyaknya berapa perbandingan yang harus dilakukan untuk mencari? $O(n)$

Sequential search (ada/tidak?)

```
// Keluaran: 1 jika ada
// atau 0 jika tidak ada

int sequentialSearch(LIST t[], int n, int val) {
    int i, last=0;

    for(i=0; i<n; i++)
       if(val==t[i].val) return 1; /* ditemukan */

    return 0; /* tidak ditemukan */
}
```


Sequential search (kemunculan pertama)

```
// Keluaran: indeks kemunculan pertama jika ada
// atau bernilai -1 jika tidak ada

int sequentialSearch(LIST t[], int n, int val) {
    int i;

    for(i=0; i<n; i++)
       if(val==t[i].val) return i; /* ditemukan */

    return -1; /* tidak ditemukan */
}
```

- Ubah fungsi `sequentialSearch` untuk melakukan variasi pencarian berikut:
  1. Indeks kemunculan terakhir
  2. Indeks seluruh kemunculan
  3. Frekuensi kemunculan x
  4. Frekuensi nilai yang lebih kecil dari x

Latihan

- Ubah fungsi `sequentialSearch` untuk melakukan variasi pencarian berikut:
  - Frekuensi kemunculan x
  - Indeks seluruh kemunculan
  - Indeks kemunculan terakhir
  - Frekuensi nilai yang lebih kecil dari x

Sequential sorted array search

- Data harus terurut
- Setiap elemen dalam *array* ditelusuri satu per satu dari awal (kiri) hingga ketemu atau nilai yang dibaca lebih besar dari nilai yang dicari atau semua elemen telah dibandingkan.
  - Asumsi: terurut *ascending*
  - Jika terurut *descending*, sesuaikan algoritme
- Mudah diimplementasikan.
- Waktu proses relatif cepat untuk kasus data yang dicari terletak di bagian depan.
  - Masih masuk lingkup $O(n)$

Sequential sorted array search

```
// Keluaran: indeks kemunculan pertama jika ada
// atau bernilai -1 jika tidak ada
// Asumsi: array t terurut secara ascending (menaik)

int sortedSearch(LIST t[], int n, int val) {
    int i=0, hsl=-1;

    while ((t[i].val<=val) && (i<n) && (-1==hsl)) {
       if(t[i].val==val) hsl=i;
       i++;
    }
    return hsl; /* tidak ditemukan */
}
```


Binary search

- Data harus terurut
- Pencarian dimulai dengan memeriksa elemen yang terletak di indeks tengah-tengah (mis. *mid*)
- Jika nilai yang di tengah lebih besar dari x, pencarian diarahkan ke data yang ada di kiri *mid*. Sebaliknya, diarahkan ke sebelah kanan.
  - Asumsi: terurut *ascending*
  - Jika terurut *descending*, sesuaikan algoritme
- Periksa elemen yang di tengah bagian tersebut
- Ulangi hingga x ditemukan atau data sudah habis
- Efisien: $O(\log n)$

Binary search

```
int binarySearch(LIST t[], int n, int val) {
    int left, right, mid;

    left = 0; right = n-1; //indeks paling kiri dan kanan
    while(left<=right) {
        mid = (left+right)/2;
        if (val<t[mid].val) //kanan tidak perlu dicek
           right = mid-1;
        else if (val>t[mid].val) //kiri tidak perlu dicek
           left = mid + 1;
        else
           return mid; /* ditemukan */
    }
    return -1; /* tidak ditemukan */
}
```


Sorting (Pengurutan)

- Mengurutkan data berdasarkan kunci tertentu.
- Jenis *sorting*:
  - *Ascending* (menaik)
  - *Descending* (menurun)
- Manfaat: mempercepat dan memudahkan akses data
- Banyak algoritme *sorting* dikembangkan untuk:
  - Mempercepat proses
  - Mengefisienkan penggunaan sumberdaya
- Ingat syarat terurut pada *binary search*

Contoh

- Masukan:

  5 2 4 6 1 3

- Keluaran:

  1 2 3 4 5 6
  (*ascending*)

  6 5 4 3 2 1
  (*descending*)

Data untuk *sorting*

- Anggaplah kita mempunyai suatu *array* integer
- Akan dilakukan *sorting* secara *ascending*
- *Sorting* secara *descending* dapat dilakukan dengan penyesuaian pada algoritme

Fungsi pendukung

- Mempertukarkan nilai t1 dan t2:

```
void tukar(LIST *t1, LIST *t2) {
   LIST t = *t1;
   *t1 = *t2;
   *t2 = t;
}
```

- Menampilkan n buah data

```
void printlist(LIST t[], int n) {
   int i;
   for (i=0; i<n; i++)
      printf("%d ", t[i].val);
   printf("\n");
}
```


BUBBLE SORT

- Ide: nilai yang besar seperti "gelembung" yang akan "naik"
- Keuntungan:
  - Sederhana dan mudah diimplementasikan
  - Cepat jika data masukan sudah terurut
- Kelemahan:
  - Secara umum membutuhkan waktu proses lebih lama
  - **Biasa dijadikan contoh algoritme pengurutan yang tidak efisien ($O(n^2)$)**

Cara kerja

- Baca elemen *array* dari kiri ke kanan, bandingkan nilai key yang bersebelahan. Jika dua nilai posisinya tidak sesuai, tukar tempatnya. Ulangi prosedur ini sampai semua elemen terurut.
- Pada putaran ke-i, nilai ke-i terbesar akan berada di posisi yang benar
  - Putaran pertama akan menempatkan nilai paling besar di akhir *array*
  - Dimanfaatkan untuk optimasi *inner loop* agar tidak lagi memeriksa nilai yang telah benar posisinya

BUBBLE SORT

```
void bubbleSort(LIST x[], int n) {
  int i,j;
  for(i=0;i<n-1;i++){
    for(j=0;j<n-(i+1);j++){
      if (x[j].val > x[j+1].val)
        tukar(&x[j],&x[j+1]);
        printf("%d %d |", i, j);
        printlist(x, n);
    }
  }
}
```

Bagian yang **ditebalkan** adalah optimasi *inner loop*

Bagian berwarna biru hanya untuk *tracing*

Algoritme *sorting* lainnya

- *Insertion sort*
- *Selection sort*
- *Quicksort*
- *Mergesort*

Review

- Untuk menampung data nama dan usia, dibuat dua variabel array:

```
char nama[SIZE][50];
int usia[SIZE];
```

- Bisakah disimpan dalam satu variabel array biasa? → TIDAK BISA, karena berbeda tipe.
- Beberapa variabel dapat dikemas dalam satu "paket" dengan struct.