Pembahasan Soal Ujian Tengah Semester Mata Kuliah

Algoritma Pemrograman UNDIP (2023)

A. Sebelum Ujian

Dengan paradigma prosedural, kompleksitas dipecah menjadi tahapan yang jelas, membuktikan bahwa setiap masalah dapat diselesaikan dengan pendekatan yang sistematis.

Sebelum Mengerjakan Ujian : Pastikan

1. Membaca dengan Jelas SETIAP PETUNJUK yang diberikan dalam soal, perhatikan apa yang diminta soal, soal Tracing berbeda dengan soal membuat program, dan subprogram tidak sama dengan program utama
2. Paradigma pemrograman prosedural berbeda dengan Paradigma pemrograman fungsional, dalam prosedural-kita diperbolehkan menggunakan fungsi yang telah didefinisikan dan dispesifikasikan (dalam kamus)

B. Ujian di Mulai

 # Soal No 1. (15%)

Diberikan potongan teks algortima di bawah ini, pada akhir eksekusi **berapa nilai akhir variable jum**, **jawaban anda disertai step by stepnya** untuk mendapatkan nilai akhir variabel jum tersebut

```
{ kamus }
    i, k, jum : integer

{ Algoritma }
   i ← 1
   jum ← 0
   while ( i <= 4 ) do
            if ( ( i mod 2 ) = 0) then
                  k traversal [ 1 . . . i ]
                           jum ← jum + i + k
            else { ( i mod 2 ) != 0 } { i ganjil }
                  jum ← jum + i + 1
            i ← i + 1
   { EndWhile }
```

EndSoal

Pada soal ini kita diminta untuk melakuikan Tracing manual, menelusuri eksekusi kode untuk memahami bagaimana program berjalan langkah demi langkah.

Program ini memulai dengan i = 1 dan jum = 0. Selama i kurang dari atau sama dengan 4, program akan mengecek apakah i genap atau ganjil.

Jika i ganjil, maka jum hanya bertambah dengan i + 1.
Jika i genap, maka ada perulangan tambahan (k traversal [1 … i]), di mana dalam setiap iterasi, jum bertambah dengan i + k.

Proses ini terus berulang hingga i = 4, lalu program berhenti dengan hasil jum = 39

Dalam proses tracing**, tidak terdapat sintaksis baku atau aturan khusus** yang harus diikuti secara ketat. Yang paling penting dalam melakukan tracing adalah menyajikan langkah-langkah eksekusi program dengan cara yang jelas, sistematis, dan mudah dipahami oleh pembaca

Tetapi kita menyarankan untuk menggunakan alur tracing berikut

```
i = 1
jum = 0
{ Memulai perulangan While }
*  Cek i {1} <= 4 ? True
   Cek i {1} mod 2 = 0 ? False
   { Masuk Kondisi else }
    jum = jum { 0 } + i {1} + 1 = 2
    i = i {1} + 1 = 2

*  Cek i {2} <= 4 ? True
   Cek i {2} mod 2 = 0 ? True
    { Memulai Loop K Traversal }
      k = 1
           jum = jum {2} + i {2} + k {1} = 5
      k = 2
           jum = jum {5} + i {2} + k {2} = 9
    { Akhir Loop K Traversal }
    i = i {2} + 1 = 3

*  Cek i {3} <= 4 ? True
   Cek i {3} mod 2 = 0 ? False
   { Masuk Kondisi else }
   jum = jum {9} + i {3} + 1 = 13
   i = i {3} + 1 = 4

*  Cek i {4} <= 4 ? True
   Cek i {4} mod 2 = 0 ? True
   { Memulai Loop K Traversal }
      k = 1
          jum = jum {13} + i {4} + k {1} = 18
      k = 2
           jum = jum {18} + i {4} + k {2} = 24
      k = 3
           jum = jum {24} + i {4} + k {3} = 31
      k = 4
           jum = jum {31} + i {4} + k {4} = 39
   { Akhir Loop K Traversal }
   i = i {4} + 1 = 5

*   Cek i {5} <= 4 ? False
   { Perulangan While Selesai }
```

Sehingga hasil Akhir dari jum adalah 39, jum = 39

Sekali lagi, Tidak ada aturan baku dalam Tracing, gaya penulisan dibebaskan, tetapi :

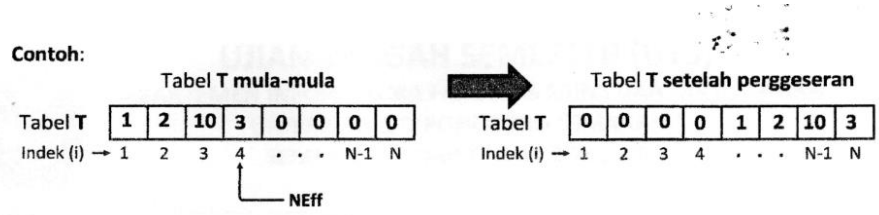
1. Penulisan Tracing harus tetap memuat semua perubahan variable yang terjadi
2. Penulisan Tracing harus dimulai **dari kondisi awal yang diberikan** dan pastikan **berakhir setelah semua perintah algoritma selesai dijalankan** dan tidak ada yang tersisa
3. Penulisan Tracing harus konsisten, perhatikan indentasi penulisan
4. Jika diperlukan, tambahkan keterangan atau catatan tambahan untuk memperjelas maksud dari setiap perubahan nilai variable

Cobalah untuk melakukan Tracing dengan Algoritma yang akan kita tulis pada bagian soal selanjutnya, untuk soalnya buatlah sampel parameter aktual sendiri sebagai Latihan

---

 # Soal No 2. (30%)

Diberikan sebuah tabel integer T dengan ukuran 1 s/d N. Tabel T berisi elemen mulai dari posisi ke 1 s/d NEff (dimana Neff <= N , dan elemen pada posisi yang belum terisi atau setelah Neff s/d N di set dengan nilai 0). Buatlah teks algoritma **dalam bentuk program utama** untuk melakukan pergeseran elemen-elemen tabel T dengan cara **meletakan elemen posisi NEff pada posisi N**, NEff-1 pada posisi N-1, Neff-2 pada posisi N-2, ...dst. Kemudian nilai elemen pada poisisi sisa dari hasil pergeseran di set menjadi 0



EndSoal

Soal kedua mengharuskan kita menyusun algoritma dalam program utama. **program utama tidak memiliki parameter input**, atau dalam kata lain tidak ada variabel yang langsung bernilai. Akibatnya, tabel T pun tidak bernilai dan belum bisa diproses secara langsung (operasi penggeseran tidak dapat dilakukan, karena isi tabel belum ada). sebelum kita dapat melakukan pemrosesan pada tabel, kita harus terlebih dahulu mendeklarasikan semua variabel yang dibutuhkan dalam soal. Setelah itu, **setiap variabel harus diinisialisasi melalui operasi input** sesuai dengan kebutuhan, sehingga data yang diperlukan tersedia dan dapat digunakan dalam proses selanjutnya

```
Program GeserTabel
   { Traversal untuk mengisi isi tabel, dengan
   membaca nilai setiap elemen dari keyboard hingga
   sebuah indeks NEff, kemudian tabel akan digeser
```

```
   menuju hingga indeks N dengan elemen sisa ditulis
   dengan 0 }
```

Catatan : teks Algoritma **selalu** terdiri dari 3 bagian : Judul, Kamus dan Algoritma, Ingat bahwa soal kali ini meminta sebuah program, oleh karena itu kita menggunakan judul 'Program', untuk Nama program dan Spesifikasi dikembalikan pada bahasa kalian - dengan syarat penulisannya harus jelas dan dapat mewakili algoritma yang akan dibuat secara umum

```
KAMUS
   constant Nmax : integer = 1000
   Neff : integer
   N : integer
   T : array [ 1 … Nmax ] of integer
   i,j,k,l,m, … : integer { sesuaikan kebutuhan untuk
   perulangan traversal(counter) }
```

Catatan : Perhatikan bahwa **array harus memiliki ukuran yang jelas**, dalam konteks ini, sebuah Array tidak dapat dituls dengan ukuran Array [1 … N] karena variabel N masih belum jelas nilainya, Solusi lain dalam masalah ini adalah kita dapat menuliskan langsung array [1 … 1000] tanpa menggunakan variabel bantuan, penulisan ini lebih aman daripada hanya menuliskan array of integer saja tanpa mendefinisikan ukuran dari tabel yang akan digunakan

```
ALGORITMA
       /. . . di tulis di sini
```

Sebelum masuk ke Penulisan algoritma, kita harus mengerti konsep dari soal yang akan kita kerjakan, dalam soal ini, terdapat dua buah indeks penting dalam tabel T, yaitu NEff dan N, dimana Neff <= N, awalnya Tabel T akan diisi dari indeks pertama (**dalam Notasi prosedural, indeks tabel dimulai dari 1**) hingga indeks ke-Neff dan untuk semua indeks dari indeks ke-NEff+1 hingga indeks ke-N akan diisi dengan 0

Perhatikan bahwa yang akan dilakukan proses input hanya dari indeks ke-1 hingga indeks ke NEff, bukan hingga indeks ke N, perhatikan juga bahwa bahwa indeks ke-Neff+1 hingga indeks ke N harus kita **inisiasi manual dengan nilai 0**

Lanjut ke Langkah penyelesaian masalah, misalkan tabel T telah terdefinisi, akan terdapat banyak cara menggeser indeks ke NEff menuju N, dan seterusnya, dua diantaranya adalah

- Geser satu satu : nilai pada indeks NEff akan digeser menuju indeks ke NEff +1, NEff+2, dan seterusnya secara sekuensial satu persatu hingga berada pada indeks ke-N, proses ini akan dilakukan sebanyak N-NEff kali, untungnya kita dapat memanfaatkan looping traversal dalam proses penggeseran ini, namun, perlu diperhatikan bahwa kita tidak hanya menggeser indeks NEff saja, indeks NEff – 1, NEff – 2 , dan seterusnya juga harus kita geser hingga menyentuh berturut-turut indeks N-1, N-2, dan seterusnya, lagi lagi dalam hal ini, kita

tetap dapat memanfaatkan looping traversal sebanyak NEff (Penggeseran dapat dilakukan dengan looping bersarang)

- Loncat : dengan Solusi Geser satu satu, kita dapat menemukan pola bahwa, nilai pada indeks ke NEff akan diletakkan pada indeks ke N, NEff-1 akan diletakkan pada indeks ke N-1, dan seterusnya hingga pada nilai indeks ke-1, akan diletakkan pada indeks ke-N-NEff+1, (Kenapa? Coba Temukan sendiri alasannya dengan membuat ilustrasi)

Perhatikan bahwa dalam setiap Solusi, nilai pada indeks ke-i akan saling menimpa, tukarlah nilai secara sekuensial, sehingga tidak ada nilai yang hilang sebelum diproses, pastikan bahwa tidak akan terjadi pemanggilan ***indeks out of range,*** Poin penting lainnya adalah, kita perlu melakukan inisiasi dengan nilai 0, untuk indeks ke 1, hingga indeks ke-N-Neff akibat penggeseran

```
ALGORITMA { geser satu satu }

  input (NEff)
  input (N)
  i traversal [ 1 … NEff ]
   { Proses : Mengisi tabel dari indeks ke-1 hingga
   indeks ke – NEff }
        input ( Ti )
  j traversal [ NEff+1 . . . N ]
   { Proses : Menginisiasi tabel dari indeks ke-
   NEff+1 hingga indeks ke – N }
        Tj ← 0

  {Proses : Mulai penggeseran elemen ke belakang }
  l traversal [NEff … 1]
       k traversal [1 ... N-Neff]
            Tk+1← Tl
  {Proses : Geser elemen satu posisi ke kanan }

  {Proses :  Set elemen awal menjadi 0 setelah
  pergeseran }
  m traversal [1 … N - NEff]
        Tm ← 0
```

Catatan : dalam Traversal i traversal [x … y], jika x > y, maka i++, tetapi jika x < y, maka i--.

```
ALGORITMA { Loncat }

  input (NEff)
  input (N)
  i traversal [ 1 … NEff ]
   { Proses : Mengisi tabel dari indeks ke-1 hingga
   indeks ke – NEff }
```

```
        input ( Ti )
  j traversal [ NEff+1 . . . N ]
   { Proses : Menginisiasi tabel dari indeks ke-
   NEff+1 hingga indeks ke – N }
        Tj ← 0

  { Proses : Pindah elemen langsung ke posisi
  akhirnya }
  k traversal [NEff … 1]
    TN-NEff+k ← Tk

  {Proses :  Set elemen awal menjadi 0 setelah
  pergeseran }
  l traversal [1 … N - NEff]
        Tl ← 0
```

Kedua Solusi di atas benar, tetapi Solusi loncat akan dinilai lebih efisien, soal nomor 2 ini menilai bagaimana logika kita berjalan untuk menyelesaikan suatu masalah yang kita bangun sendiri dari 0 (tidak ada parameter input)

Kedua Solusi di atas ekuivalen dengan kode program (bahasa C) berikut :

```c
// Solusi Geser satu Satu
    scanf("%d", &N);
    scanf("%d", &NEff);

    // Input nilai tabel
    for (i = 1; i <= NEff; i++) {
        scanf("%d", &T[i]);
    }
    // Inisialisasi elemen setelah NEff menjadi 0
    for (j = NEff + 1; j <= N; j++) {
        T[j] = 0;
    }
    // Proses geser satu per satu
    for (l = NEff ; l >= 1; l--) {
        for (k = 1; k <= N-NEff; k++) {
            T[k+l] = T[l];
        }
    }
    // Isi elemen sisa dengan 0
    for (m = 1; m <= (N - NEff); m++) {
        T[m] = 0;
    }
```

```c
// Solusi Loncat
    scanf("%d", &N);
    scanf("%d", &NEff);

    // Input elemen tabel
    for (i = 1; i <= NEff; i++) {
```

```
        scanf("%d", &T[i]);
    }
    // Inisialisasi elemen setelah NEff menjadi 0
    for (i = NEff + 1; i <= N; i++) {
        T[i] = 0;
    }
    // Proses loncat langsung ke posisi akhir
    for (k = NEff; k >= 1; k--) {
        T[N - NEff + k] = T[k];
    }
    // Set elemen awal menjadi 0 setelah pergeseran
    for (i = 1; i <= (N - NEff); i++) {
        T[i] = 0;
    }
```

 # Soal No 3. (25%)

Diberikan sebuah *type* TabInt : array [1..100] of integer . Buatlah teks algoritma dalam bentuk subprogram untuk fungsi di bawah ini

```
Function IsTabelSimetri(T1 : TabInt, T2 : TabInt) → Boolean
{ mengirimkan TRUE jika nilai setiap elemen T1 sama dengan T2
}
{ Tabel T1 dan T2 selalu memiliki Panjang sama, sehingga
tidak perlu dilakukan pengecekan Panjang T1 dan T2 }
{ contoh: }

{ T1 = < 1 3 4 5 7 8 9 >, T2 = < 1 3 4 5 7 8 9 > maka TRUE }
{ T1 = < 1 3 4 5 7 8 1 >, T2 = < 1 3 4 5 7 8 9 > maka FALSE }

{ Kamus Lokal }

{ Algoritma }
```

EndSoal

Sekali lagi, soal ini akan menguji kompleksitas algoritma kalian, dalam soal, jelas ditulis bahwa Tabel T1 dan T2 selalu memiliki Panjang yang sama dan **ukuran tabel pasti 100,** informasi ini sangat berguna untuk menentukan algoritma yang akan kita buat

Karena ukuran tabel selalu 100 dan panjangnya sama, kita dapat langsung membandingkan elemen satu per satu dengan hanya satu *looping* traversal tanpa perlu melakukan pengecekan tambahan terkait panjang tabel. Dengan pendekatan ini, kita cukup melakukan iterasi dari indeks pertama hingga terakhir (100) dan mengevaluasi kesamaan setiap elemen T1 dan T2. Jika ada **satu saja** elemen yang berbeda, fungsi dapat langsung mengembalikan **FALSE** tanpa harus mengecek elemen lainnya, sehingga algoritma lebih optimal

Berikut ini adalah Solusi Optimal Tanpa Nama Antara

```
{ Kamus Lokal }
```

```
    i : integer
```

```
{ Algoritma }
    i traversal [1 … 100 ]
            if T1ᵢ ≠ T2ᵢ then
                → FALSE
    → TRUE
```

Berikut ini adalah Solusi Optimal dengan Nama Antara

```
{ Kamus Lokal }
    i : integer
    hasil : boolean
```

```
{ Algoritma }
    Hasil ← TRUE
    i traversal [1 … 100 ]
            if T1ᵢ ≠ T2ᵢ then
                hasil ← FALSE
    → hasil
```

Tidak terdapat perbedaan yang signifikan antara kedua algoritma di atas, hanya gaya penulisannya yang berbeda. Algoritma pertama lebih ringkas dan langsung memberikan hasil tanpa variabel tambahan, sedangkan algoritma kedua menggunakan variabel hasil untuk menyimpan status perbandingan sebelum dikembalikan sebagai output. Meskipun keduanya memiliki kompleksitas yang sama, penggunaan variabel antara pada algoritma kedua membuatnya lebih aman.

Berikut ini adalah solusi kurang effisien yang tidak disarankan

```
{ Kamus Lokal }
    i : integer
    function getSize(T : TabInt) → integer
    { getSize(T) mengembalikan ukuran dari tabel T yang
    terdefinisi }
```

```
{ Algoritma }
    if getSize(T1) ≠ getSize(T2) then
            → FALSE
    else
          i traversal [1 … 100 ]
                if T1ᵢ ≠ T2ᵢ then
                    → FALSE
          → TRUE
```

Catatan : sebuah subprogram dapat ditulis di kamus dan dapat langsung digunakan dalam algoritma tanpa harus merealisasikannya asalkan subprogram dispesifikkasikan dengan jelas sebelumnya

Sebenarnya Solusi di atas dapat dikatakan aman, namun karena dalam soal telah ditegaskan ukuran masing masing tabel, maka kita wajib mendesain algoritma sesuai dengan permintaan

---

 # Soal No 4. (30%)

Buatlah teks algoritma dalam bentuk **program utama** untuk menghitung banyaknya kata yang diakhiri dengan **pasangan karakter 'LE'** dari sebuah pita karakter. Definisi kata yang digunakan pada persolan ini adalah **jika ketemu spasi, koma, dan titik**. Anda **dapat langsung menggunakan** primitif Mesin Karakter (START(), ADV(), dan EOP()) tanpa harus melakukan realisasi.

Contoh. Pita karakter='Sa**le** pisang lebih enak dibandingkan ikan le**le**, punya Su**le**.'

Dari pita karakter diatas akan menghasilkan 3 kata yang diakhiri dengan pasangan 'LE'

#EndSoal

Selamat berkreasi ! Selamat datang di Mesin abstrak ini !

Mesin karakter adalah konsep mesin abstrak yang terdiri dari beberapa komponen utama:

- Pita yang berisi deretan karakter dan diakhiri dengan tanda titik (.). Jika pita hanya berisi titik, maka disebut sebagai pita kosong.
- Tombol START dan ADV yang digunakan untuk mengontrol pergerakan mesin.
- Lampu EOP (End Of Pita) yang menyala (Boolean : True) ketika mesin mencapai akhir pita, yaitu saat karakter di jendela adalah titik.
- Jendela karakter, yaitu bagian dari mesin yang hanya dapat menampilkan satu karakter pada suatu waktu. Karakter yang berada dalam jendela disebut sebagai CC (Current Character) dan hanya karakter ini yang dapat dibaca, sedangkan karakter lain di pita tidak terlihat.

Mesin ini memiliki mekanisme untuk menggeser posisi pita dan mendeteksi akhir pita dengan menyalakan lampu EOP jika karakter yang dibaca adalah titik. Kondisi atau state mesin setiap saat ditentukan oleh nilai CC dan status lampu EOP. Mesin hanya dapat beroperasi jika lampu EOP belum menyala

Perlu diperhatikan bahwa pita karakter mungkin kosong. Dalam soal ini, tentu saja ada beberapa ide yang memberikan Solusi akhir yang sama , misalkan SumLe adalah variabel yang akan kita pakai sebagai *counter* LE dalam soal ini, solusi solusi tersebut diantaranya :

1. Langsung bandingkan tiga buah karakter bersebelahan sekaligus, ambil 3 buah karakter yang bersandingan secara sekuensial, mula mula ambil karakter ke-1, ke-2 dan ke-3, kemudian cek ketiganya, jika karakter ke-1 adalah ‘L/l’ , karakter ke-2 adalah ‘E/e’, dan karakter ke-3 adalah ‘./ /,’, SumLe bertambah 1, kemudian majukan lagi satu karakter, cek karakter ke-2,ke-3,dan ke-4 secara bersamaan, program akan berhenti jika CC ≠ ‘.’, perhatikan bahwa karakter yang

dibandingkan bukanlah CC, melainkan variabel baru yang dapat kita misalkan sebagai CC1, CC2 dan CC3

```
Program CounterLE-1
   { mencari karakter 'L', 'E' dan "./ /," yang
berturut turut bersebelahan }
```

```
KAMUS
   SumLe : integer
   CC1, CC2, CC3 : character
```

```
ALGORITMA
   START()
   SumLe ← 0
   CC1 ← ' '  { CC1 dan CC2 belum bernilai }
   CC2 ← ' '
   CC3 ← CC
   repeat{ not EoP }
       if ((CC1 = 'L') OR (CC1 = 'l')) AND
          ((CC2 = 'E') OR (CC2 = 'E')) AND
          ((CC3 = ' ') OR (CC3 = ',') OR (CC3 = '.')
       then
           SumLe ← SumLe + 1
       CC1 ← CC2
       CC2 ← CC3
       ADV()
       CC3 ← CC
   until(CC = '.')

   if ((CC1 = 'L') OR (CC1 = 'l')) AND
      ((CC2 = 'E') OR (CC2 = 'E')) AND
      (CC3 = '.')then
   SumLe ← SumLe + 1


   output(SumLe)
```

catatan : Solusi ini merupakan modifikasi dari solusi pada buku "Draft Diktat Kuliah Dasar Pemrograman (Bagian Pemrograman Prosedural)" oleh Inggriani Liem pada halaman 166, pengecekan terakhir harus dilakukan karena dalam perulangan di atas, ketika CC3 mencapai ujung (karakter '.') maka operasi kondisional dalam perulangan tidak akan dilakukan karena kondisi akhir terpenuhi, oleh karena itu , kita menggunakan kondisional tambahan di akhir

Dengan contoh yang diberikan berikut adalah alur pengerjaannya jika menggunakan algoritma ini :

'Sale pisang lebih enak dibandingkan ikan lele, punya Sule.'
'Sale pisang lebih enak dibandingkan ikan lele, punya Sule.'

'Sale pisang lebih enak dibandingkan ikan lele, punya Sule.'
'Sale pisang lebih enak dibandingkan ikan lele, punya Sule.'
'Sale pisang lebih enak dibandingkan ikan lele, punya Sule.' [Ditemukan]
'Sale pisang lebih enak dibandingkan ikan lele, punya Sule.'

….

Solusi ini mirip seperti kita menghitung sebuah karakter (missal CounterCharA) bedanya, kita langsung mencari seperangkat karakter, bukan karakter Tunggal

2. Solusi kedua adalah : mencari karakter 'L' terlebih dahulu, kemudian cocokkan apakah karakter setelah 'L' ini adalah 'E', jika iya cocokkan lagi karakter setelahnya adakah karakter setelah 'E' adalah'./ /.' , jika iya, SumLe bertambah 1

```
Program CounterLE-2
   { mencari karakter 'L'kemudian membandingkan
karakter setelahnya hingga terkumpul karakter "LE."
Atau "LE " atau "LE," yang bersebelahan }
```

```
KAMUS
   SumLe : integer
```

```
ALGORITMA
   START()
   SumLe ← 0
   While(CC ≠ '.') do
      { cari hingga ditemukan 'L'}
      While( CC ≠ 'L' AND CC ≠ '.') do
           ADV()
      { EndWhile, berarti CC = 'L' atau '.' }
      if(CC = 'L') then
           ADV()
           If(CC = 'E') then
                ADV()
                If(CC = ' ' OR CC = ',' OR CC = '.')
                then
                     SumLe = SumLe + 1


   output(SumLe)
```

catatan : dengan Solusi ini, kita tidak perlu lagi melakukan pengecekan di akhir pita, karena algoritma ini memastikan bahwa karakter sebelum '.' Bukanlah 'E'

C. End of File

- Tujuan utama dari pembahasan ini adalah untuk memperdalam konsep algoritma dan pemrograman secara mandiri.
- Tidak ada unsur plagiarisme atau pelanggaran etika dalam pembuatan dan pembahasan soal ini.
- Setiap solusi yang diberikan didasarkan pada pemahaman sendiri dan dapat menjadi bahan diskusi untuk meningkatkan wawasan.
- Pembahasan ini tidak dimaksudkan untuk menyinggung atau merugikan pihak mana pun.
- Jika ada kekeliruan, maka itu merupakan bagian dari proses pembelajaran dan dapat dikoreksi bersama.

"Seperti sebuah pita yang mencapai akhir barisannya, akhir bukanlah kegagalan, tetapi tanda bahwa kita telah menyusun sesuatu yang bermakna. Bahkan ketika kita mencapai End of File, kita selalu bisa memulai kembali—dengan baris kode baru, ide baru, dan semangat yang tak terbatas." — Terinspirasi dari semangat Ada Lovelace

D. About file

File Name : Pembahasan_UTS_Alpro_IFUNDIP23
Author : Aves
Created On : Kudus, 05/April/2025 : 16:39 WIB
Last Update : 06/April/2025 : 10:24 WIB