LAPORAN PRAKTIKUM

Algoritma Pemrograman

MODUL 5 & 6

FOR-LOOP

Disusun oleh:

EDWARD ABIMAS SURYA HATTA

109082500171

S1IF-13-04

PROGRAM STUDI S1 INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

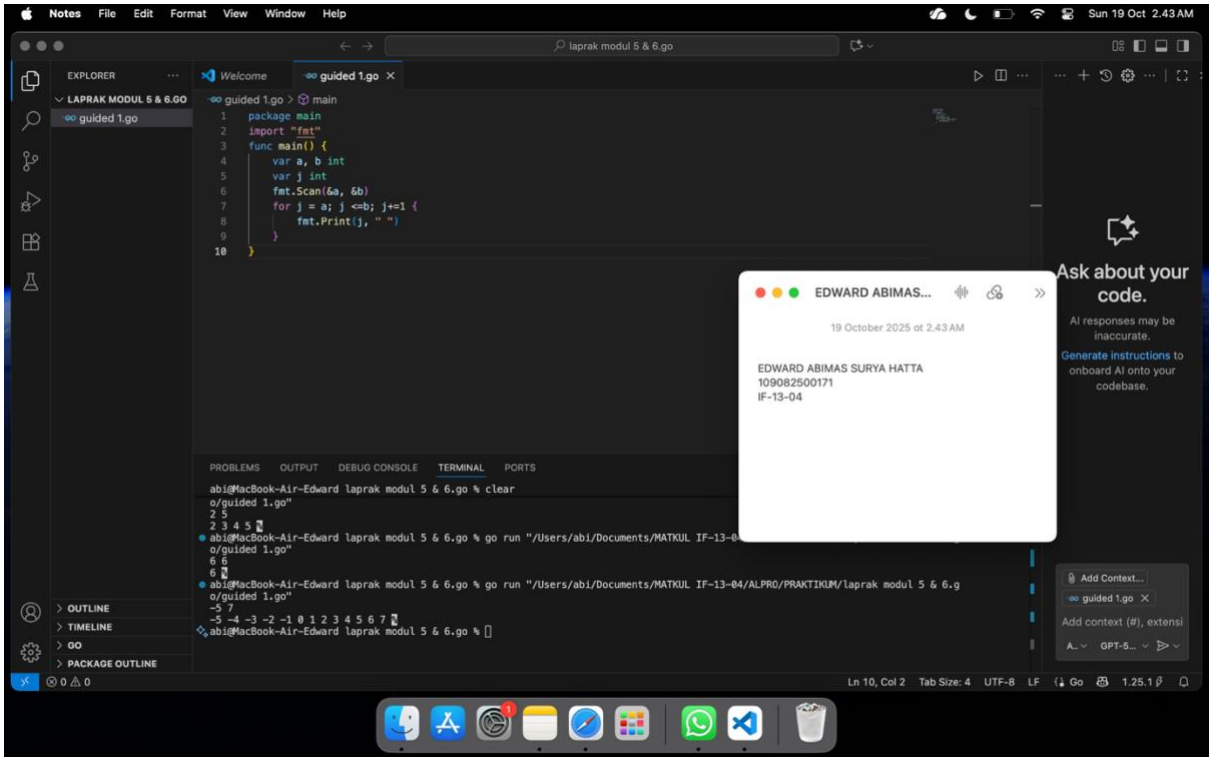
LATIHAN KELAS – GUIDED

1. **Guided 1**

Source Code

```
package main
import "fmt"
func main() {
    var a, b int
    var j int
    fmt.Scan(&a, &b)
    for j = a; j <=b; j+=1 {
        fmt.Print(j, " ")
    }
}
```

Screenshoot program



Deskripsi program

Kode ini dimulai dengan pernyataan package main. Dalam Go, ini adalah deklarasi standar yang menandakan bahwa file ini kompilasi menjadi sebuah program yang dapat dieksekusi (executable), bukan sebuah *library* yang akan digunakan oleh program lain.

Selanjutnya, kita melihat import "fmt". Perintah ini menginstruksikan kompiler Go untuk menyertakan fungsionalitas dari paket "fmt", yang merupakan singkatan dari *format*. Paket ini sangat penting karena menyediakan fungsi-fungsi untuk menangani *input* (masukan) dan *output* (keluaran) yang terformat, seperti membaca data dari pengguna atau mencetak teks ke konsol.

Kemudian, kita masuk ke blok utama program, yaitu func main(). Fungsi main adalah titik masuk (entry point) dari setiap program Go yang bisa dieksekusi. Ketika Anda menjalankan program ini, sistem operasi akan memulai eksekusi kode dari baris pertama di dalam fungsi main.

Di dalam func main, baris pertama adalah deklarasi variabel: var a, b int dan var j int. Di sini, kita mendeklarasikan tiga variabel dengan tipe data int (integer atau bilangan bulat). Variabel a dan b dimaksudkan untuk menyimpan batas awal dan batas akhir dari rentang angka, sementara j akan kita gunakan sebagai variabel *iterator* atau pencacah dalam perulangan (*looping*).

Setelah variabel siap, program menjalankan fmt.Scan(&a, &b). Ini adalah fungsi dari paket fmt yang digunakan untuk membaca *input* dari pengguna. Program akan berhenti di baris ini dan menunggu pengguna mengetikkan sesuatu di terminal. Pengguna diharapkan memasukkan dua angka bulat yang dipisahkan oleh spasi (atau *newline*). Tanda ampersand (&) sebelum a dan b adalah operator *pointer*; ini berarti kita memberikan alamat memori dari variabel a dan bke fungsi Scan, sehingga Scan dapat secara langsung mengisi nilai ke dalam variabel tersebut berdasarkan apa yang diketik pengguna.

Bagian inti dari program ini adalah perulangan for: for j = a; j <=b; j+=1. Ini adalah struktur *looping* klasik yang terdiri dari tiga komponen.

1. **Inisialisasi:** j = a. Sebelum *loop* dimulai, variabel pencacah j diatur nilainya agar sama dengan nilai a (angka pertama yang dimasukkan pengguna).
2. **Kondisi:** j <= b. *Loop* akan terus dieksekusi (beriterasi) selama kondisi ini bernilai benar (*true*), yaitu selama nilai jmasih kurang dari atau sama dengan nilai b (angka kedua yang dimasukkan pengguna).
3. **Increment (Penaikan):** j+=1. Pernyataan ini dieksekusi *setelah* setiap iterasi selesai. Ini berarti nilai j akan ditambah satu setiap kali *loop* berputar. j+=1 adalah cara singkat untuk menulis j = j + 1.

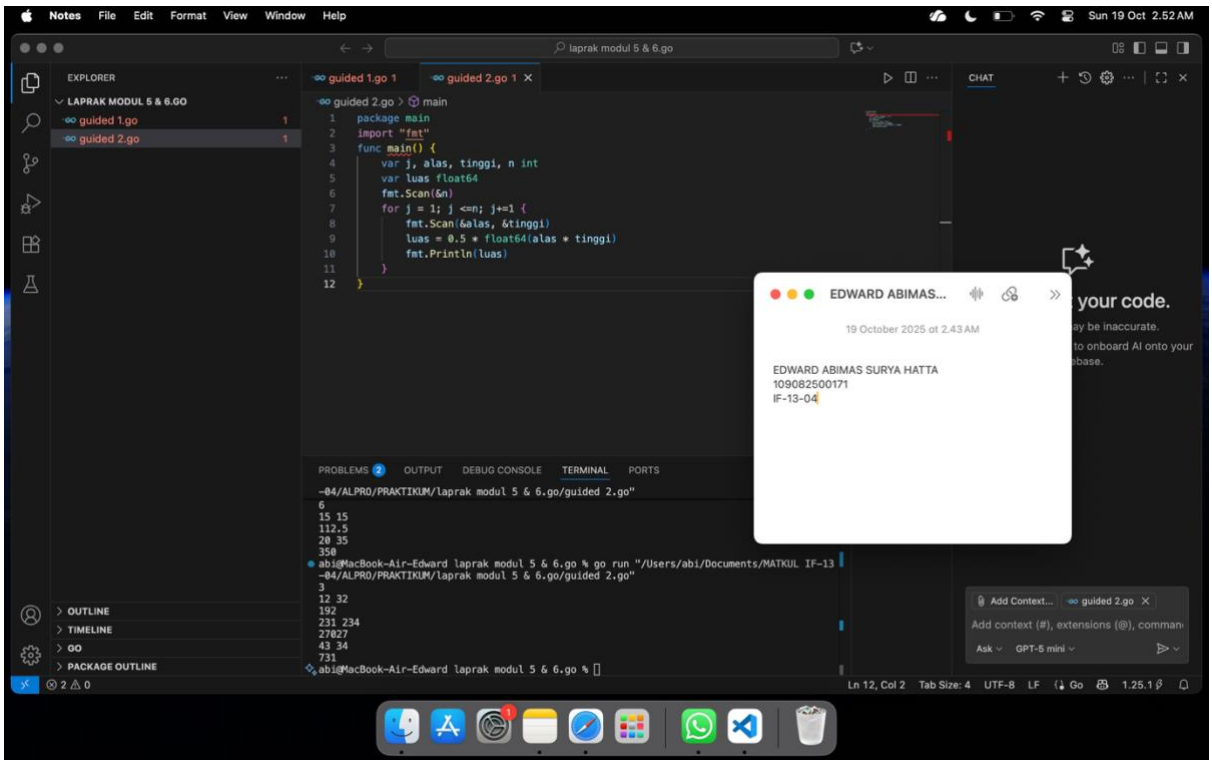
   Di dalam blok *loop* for tersebut, terdapat satu perintah: fmt.Print(j, " "). Perintah ini menggunakan fungsi Print dari paket fmtuntuk mencetak nilai j saat ini ke layar konsol, diikuti dengan satu karakter spasi (" "). Penggunaan Print (bukan Println) penting di sini, karena Print tidak menambahkan baris baru (*newline*) setelah mencetak. Ini memastikan bahwa semua angka akan dicetak secara berurutan dalam satu baris yang sama.

2. Guided 2

Source Code

```
package main
import "fmt"
func main() {
    var j, alas, tinggi, n int
    var luas float64
    fmt.Scan(&n)
    for j = 1; j <=n; j+=1 {
        fmt.Scan(&alas, &tinggi)
        luas = 0.5 * float64(alas * tinggi)
        fmt.Println(luas)
    }
}
```

Screenshoot program



Deskripsi program

Program ini dimulai dengan package main, yang menandakan bahwa ini adalah program yang dapat dieksekusi secara mandiri, dan import "fmt" untuk memasukkan pustaka (*library*) yang menangani fungsi masukan dan keluaran (input/output).
Fungsi main adalah tempat logika utama program berjalan. Di awal main, kita melihat deklarasi beberapa variabel. var j, alas, tinggi, n int mendeklarasikan empat variabel—j, alas, tinggi, dan n—sebagai tipe data int atau bilangan bulat. Variabel j akan berfungsi sebagai pencacah (iterator) untuk perulangan, n akan digunakan untuk menentukan berapa kali kita ingin mengulang proses, sedangkan alas dan tinggi akan menampung nilai masukan dari pengguna.
Selanjutnya, var luas float64 mendeklarasikan variabel luas dengan tipe data float64. Tipe data ini (bilangan desimal dengan presisi ganda) dipilih karena hasil perhitungan luas, yang menggunakan pengali 0.5, kemungkinan besar akan menghasilkan angka desimal (pecahan).
Baris pertama yang dieksekusi adalah fmt.Scan(&n). Program akan berhenti di sini dan menunggu pengguna memasukkan satu nilai bilangan bulat. Nilai ini akan disimpan ke dalam variabel n (berkat operator *pointer* &). Variabel n ini sangat penting karena ia mengontrol berapa kali perulangan for akan dieksekusi.
Setelah n didapat, program masuk ke struktur perulangan for j = 1; j <=n; j+=1. Mari kita bedah: j = 1 menginisialisasi pencacah j dengan nilai 1. j <= n adalah kondisi; perulangan akan terus berjalan selama nilai j masih kurang dari atau sama dengan nilai n yang dimasukkan pengguna tadi. j+=1 berarti setelah setiap iterasi (putaran) selesai, nilai j akan ditambah satu. Sederhananya, blok kode di dalam for ini akan dieksekusi sebanyak n kali.
Sekarang kita lihat apa yang terjadi *di dalam* setiap iterasi perulangan.
Baris pertama di dalam *loop* adalah fmt.Scan(&alas, &tinggi). Sama seperti Scan untuk n, program kembali berhenti dan menunggu masukan dari pengguna. Namun, kali ini program mengharapkan dua nilai int, yaitu untuk alas dan tinggi. Jika ntadi adalah 3, maka program akan meminta masukan alas dan tinggi ini sebanyak 3 kali secara terpisah.
Setelah mendapatkan alas dan tinggi untuk iterasi saat ini, program melakukan perhitungan: luas = 0.5 * float64(alas * tinggi). Perhitungan ini adalah inti dari program. Pertama, (alas * tinggi) dieksekusi. Karena keduanya int, hasilnya juga int. Kemudian, float64(alas * tinggi) melakukan *type casting* atau konversi tipe data. Hasil perkalian integer tadi diubah menjadi float64. Langkah ini sangat penting; jika tidak, perkalian dengan 0.5 mungkin tidak akurat atau bahkan gagal. Setelah dikonversi, nilai tersebut dikalikan dengan 0.5 (yang secara *default* dianggap sebagai float), dan hasil akhirnya disimpan ke dalam variabel luas.

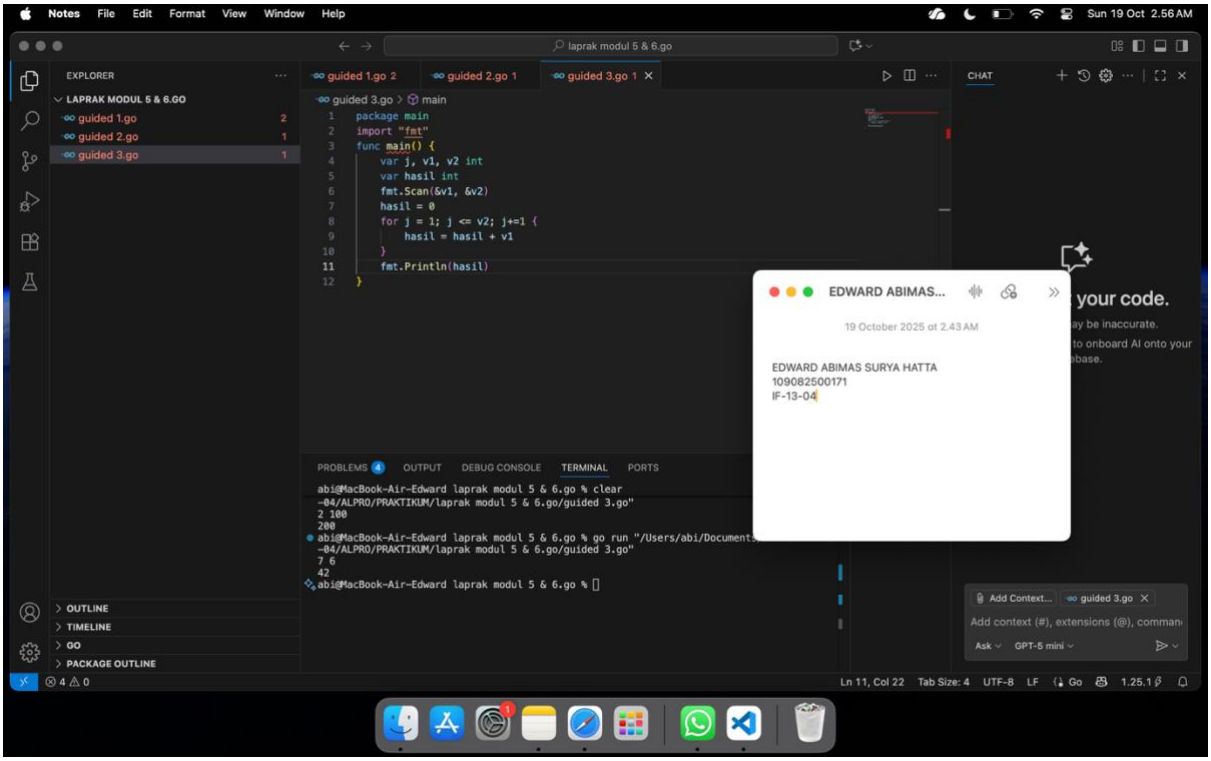
Baris terakhir di dalam *loop* adalah fmt.Println(luas). Fungsi Println (Print Line) akan mencetak nilai yang tersimpan di variabel luas ke konsol, lalu secara otomatis pindah ke baris baru.

3. **Guided 3**

Source Code

```
package main
import "fmt"
func main() {
    var j, v1, v2 int
    var hasil int
    fmt.Scan(&v1, &v2)
    hasil = 0
    for j = 1; j <= v2; j+=1 {
        hasil = hasil + v1
    }
    fmt.Println(hasil)
}
```

Screenshoot program



Deskripsi program

Program ini dimulai dengan package main, yang mendeklarasikan bahwa file ini adalah sebuah program yang dapat dieksekusi, dan import "fmt", yang memuat pustaka (library) standar Go untuk menangani *input* dan *output* (masukan dan keluaran) yang diformat.

Selanjutnya, kita masuk ke func main(), yang merupakan fungsi utama di mana eksekusi program dimulai. Di dalam fungsi ini, pertama-tama kita mendeklarasikan variabel. var j, v1, v2 int menyiapkan tiga variabel (j, v1, dan v2) yang semuanya bertipe int atau bilangan bulat. Variabel j akan kita gunakan sebagai pencacah (iterator) dalam perulangan, sementara v1dan v2 akan menampung nilai yang dimasukkan oleh pengguna. Kemudian, var hasil int mendeklarasikan variabel hasil yang juga bertipe int, yang akan kita gunakan untuk menyimpan nilai akumulasi perhitungan.

Setelah deklarasi, program menjalankan fmt.Scan(&v1, &v2). Perintah ini akan menjeda program dan menunggu pengguna memasukkan dua nilai bilangan bulat dari terminal. Nilai pertama akan disimpan ke dalam variabel v1, dan nilai kedua akan disimpan ke v2.

Langkah berikutnya adalah hasil = 0. Ini adalah langkah *inisialisasi* yang sangat penting. Kita mengatur nilai awal hasilmenjadi nol. Ini ibarat kita menyiapkan wadah yang kosong sebelum kita mulai mengisinya. Jika kita tidak mengaturnya ke nol,

variabel hasil mungkin berisi "sampah" memori atau nilai *default* yang tidak terduga, yang akan merusak perhitungan kita.

Inti dari program ini ada di perulangan for j = 1; j <= v2; j+=1. Mari kita bedah struktur ini. j = 1 adalah inisialisasi *loop*; pencacah j dimulai dari 1. j <= v2 adalah kondisi; *loop* akan terus berulang selama nilai j masih kurang dari atau sama dengan nilai v2 (angka kedua yang dimasukkan pengguna). j+=1 adalah *increment*; setelah setiap putaran *loop* selesai, nilai j akan ditambah satu. Secara efektif, blok kode di dalam for ini akan dieksekusi sebanyak v2 kali.

Di dalam setiap putaran *loop*, perintah hasil = hasil + v1 dieksekusi. Ini adalah proses *akumulasi*. Program mengambil nilai hasil saat ini, menambahkannya dengan nilai v1, lalu menyimpan nilai baru tersebut kembali ke dalam variabel hasil. Proses ini berulang terus menerus.

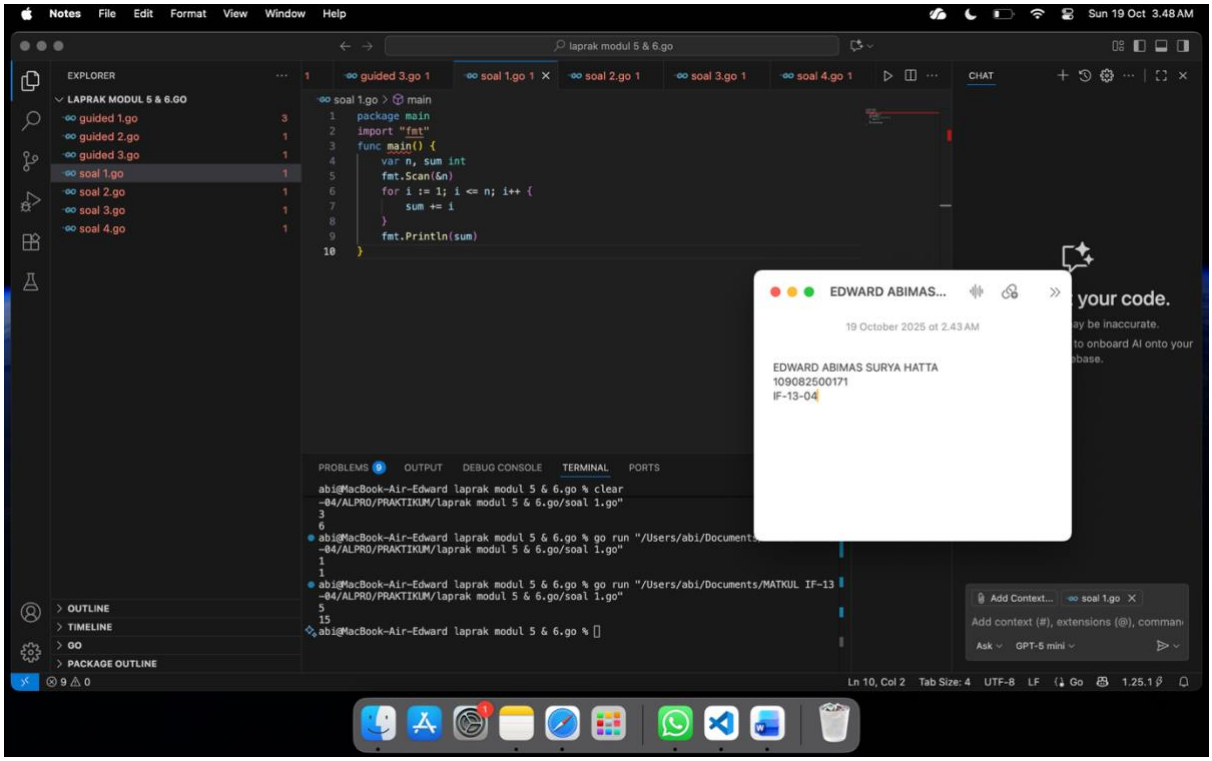
TUGAS

1. Tugas 1

Source code

```
package main

import "fmt"

func main() {

    var n, sum int

    fmt.Scan(&n)

    for i := 1; i <= n; i++ {

        sum += i

    }

    fmt.Println(sum)

}
```

Screenshoot program



Deskripsi program

Kode ini diawali dengan package main, yang merupakan deklarasi wajib untuk program Go yang bisa dieksekusi (executable). Ini memberitahu kompiler bahwa paket ini harus dikompilasi menjadi sebuah file biner yang bisa dijalankan. Selanjutnya, import "fmt" adalah perintah untuk menyertakan paket "fmt", yang berisi fungsionalitas untuk melakukan operasi *input/output* terformat, seperti membaca masukan dari konsol dan mencetak keluaran ke konsol.

Kemudian kita masuk ke func main(), ini adalah titik masuk utama program. Eksekusi kode akan dimulai dari sini.

Di dalam main, baris pertama adalah var n, sum int. Di sini kita mendeklarasikan dua variabel, n dan sum, keduanya dengan tipe data int (integer atau bilangan bulat). Penting untuk dicatat bahwa di Go, variabel yang dideklarasikan tanpa nilai awal (seperti sum di sini) akan otomatis diinisialisasi ke "nilai nol" (zero value) untuk tipenya. Untuk int, nilai nolnya adalah 0. Jadi, pada titik ini, sum sudah memiliki nilai 0. Variabel n akan kita gunakan untuk menyimpan batas atas angka yang dimasukkan pengguna, dan sum akan bertindak sebagai *akumulator* atau penampung total penjumlahan.

Baris berikutnya adalah fmt.Scan(&n). Program akan berhenti di baris ini dan menunggu pengguna mengetikkan sebuah bilangan bulat di terminal. Ketika pengguna menekan Enter, nilai tersebut akan dibaca dan disimpan ke dalam alamat memori dari variabel n (itulah fungsi dari simbol &).

Bagian inti dari program ini adalah perulangan for. Di Go, for adalah satu-satunya konstruksi *looping*. Struktur for i := 1; i <= n; i++ adalah bentuk klasik *loop* yang memiliki tiga komponen:

1. **Inisialisasi: i := 1. Ini adalah pernyataan singkat untuk mendeklarasikan *sekaligus* menginisialisasi variabel i(sebagai iterator atau**

pencacah) dengan nilai awal 1. Variabel i ini hanya "hidup" di dalam lingkup (scope) perulangan for ini.

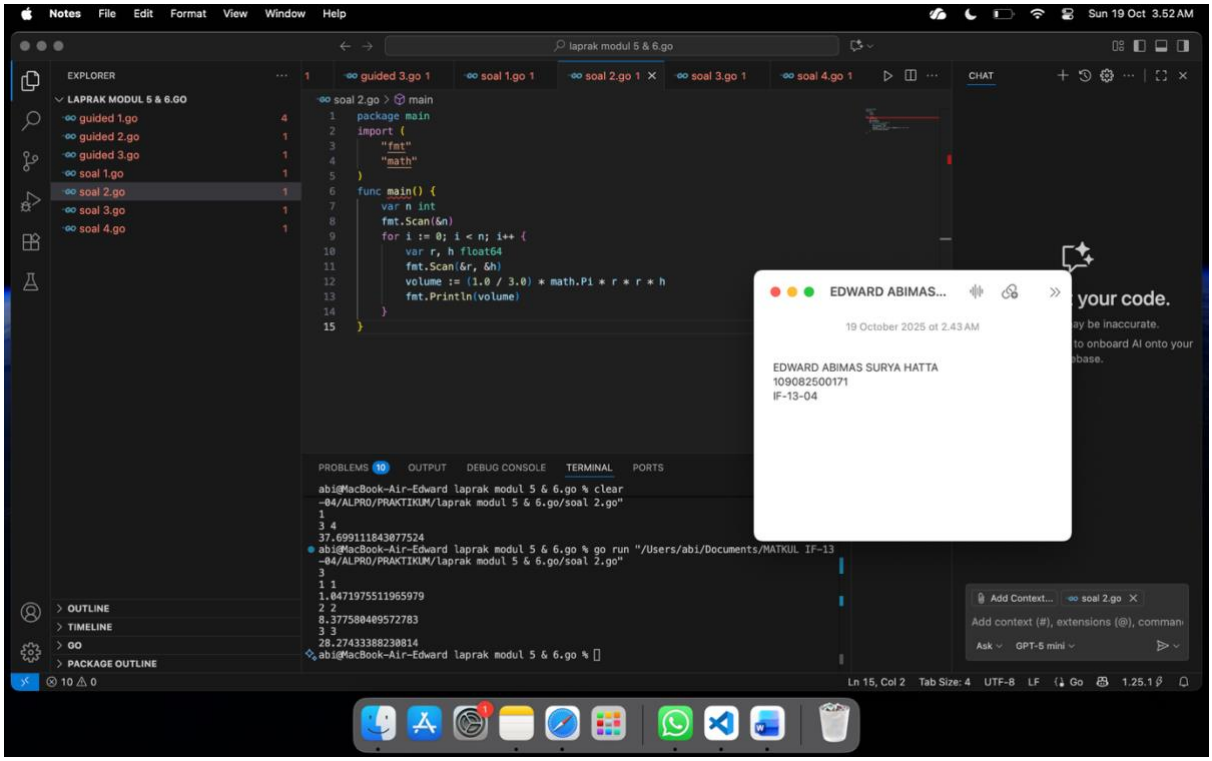
2. **Kondisi: i <= n. Sebelum setiap iterasi (putaran) dimulai, kondisi ini akan diperiksa. Jika kondisi ini bernilai benar (*true*), yaitu jika nilai i saat ini masih kurang dari atau sama dengan n (angka yang dimasukkan pengguna), maka blok kode di dalam *loop* akan dieksekusi.**
3. **Post-statement (Increment): i++. Pernyataan ini dieksekusi *setelah* blok kode di dalam *loop* selesai dijalankan untuk satu iterasi. i++ adalah singkatan dari i = i + 1, yang artinya menaikkan nilai i sebesar satu.**
   Di dalam *loop*, hanya ada satu baris: sum += i. Ini adalah operator penugasan gabungan (compound assignment) yang merupakan kependekan dari sum = sum + i. Di sinilah proses akumulasi terjadi. Setiap kali *loop* berputar, program mengambil nilai i saat itu dan menambahkannya ke nilai sum yang sudah ada.

2. Tugas 2

Source code

```
package main
import (
    "fmt"
    "math"
)
func main() {
    var n int
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        var r, h float64
        fmt.Scan(&r, &h)
        volume := (1.0 / 3.0) * math.Pi * r * r * h
        fmt.Println(volume)
    }
}
```

Screenshoot program



Deskripsi program

Program ini dimulai dengan pernyataan package main, yang menandakan bahwa file ini adalah sebuah program *executable*(bisa dieksekusi). Di bawahnya, kita melihat blok import yang kali ini menyertakan dua paket: "fmt" dan "math". Paket "fmt"kita perlukan, seperti biasa, untuk menangani fungsi *input* (masukan) dan *output* (keluaran), yaitu Scan dan Println. Paket "math" adalah paket standar Go yang kita impor karena kita membutuhkan konstanta matematika presisi tinggi, dalam hal ini math.Pi (nilai Pi,).

Selanjutnya kita masuk ke func main(), yang merupakan titik awal eksekusi program. Di dalam main, baris pertama adalah var n int. Kita mendeklarasikan sebuah variabel n dengan tipe data int (bilangan bulat). Variabel ini akan kita gunakan untuk menentukan berapa kali kita ingin melakukan perhitungan.

Setelah deklarasi, program menjalankan fmt.Scan(&n). Program akan berhenti di sini dan menunggu pengguna memasukkan satu angka bulat, yang kemudian akan disimpan ke dalam variabel n.

Setelah nilai n didapat, program masuk ke struktur perulangan for i := 0; i < n; i++. Ini adalah *loop* yang akan dieksekusi sebanyak n kali. Inisialisasinya i := 0 (membuat iterator i dimulai dari 0), kondisinya i < n (loop berjalan selama i masih di bawah n), dan *increment*-nya i++ (tambahkan i dengan 1 setelah setiap iterasi).

Sekarang, kita lihat apa yang terjadi *di dalam* perulangan. Setiap kali *loop* berputar, dua baris pertama di dalamnya akan dieksekusi: var r, h float64 dan fmt.Scan(&r, &h). Pertama, program mendeklarasikan dua variabel baru, r dan h, dengan tipe data float64 (bilangan desimal presisi ganda). Tipe float64 ini penting karena nilai

jari-jari (r) dan tinggi (h), serta hasil perhitungan volume, kemungkinan besar adalah angka desimal. Setelah variabel siap, fmt.Scan kembali meminta masukan dari pengguna, kali ini dua angka (bisa desimal), yang akan disimpan ke r dan h.

Baris inti dari logika program ini adalah volume := (1.0 / 3.0) * math.Pi * r * r * h. Mari kita bedah. volume := adalah sintaks deklarasi singkat di Go; variabel volume otomatis dibuat dengan tipe float64 karena hasil perhitungannya adalah *float*. Perhitungannya sendiri adalah rumus untuk volume kerucut, yaitu . Penting untuk diperhatikan bahwa kita menulis (1.0 / 3.0), bukan (1 / 3). Jika kita menulis (1 / 3), Go akan melakukan pembagian integer dan hasilnya adalah 0, yang akan membuat seluruh perhitungan volume salah. Dengan menulis 1.0 / 3.0, kita memaksa Go melakukan pembagian *floating-point*. math.Pi adalah konstanta Pi yang kita ambil dari paket math, dan r * r adalah kuadrat.

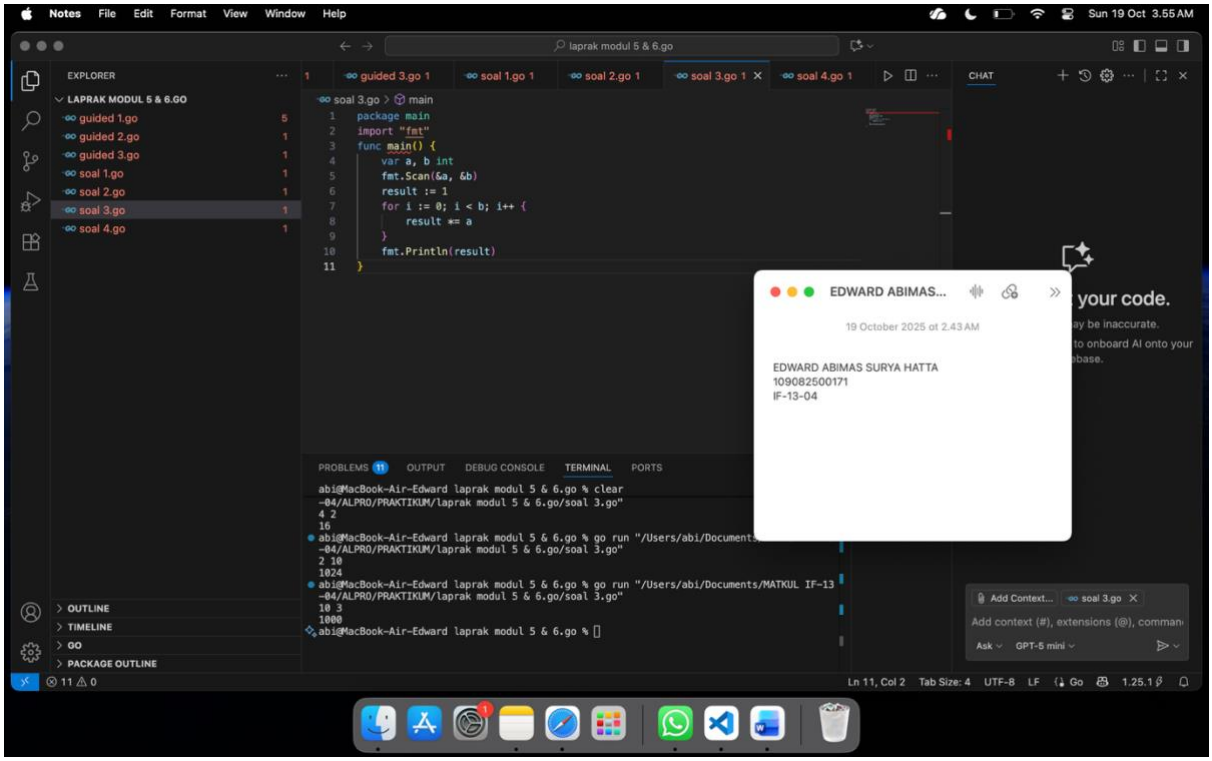
Setelah nilai volume didapat, baris terakhir di dalam *loop*, fmt.Println(volume), mencetak nilai volume tersebut ke konsol, diikuti dengan baris baru.

3. **Tugas 3**

Source code

```
package main
import "fmt"
func main() {
    var a, b int
    fmt.Scan(&a, &b)
    result := 1
    for i := 0; i < b; i++ {
        result *= a
    }
    fmt.Println(result)
}
```

Screenshoot program



Deskripsi program

Program ini, seperti biasa, dimulai dengan package main, yang menandakan bahwa ini adalah sebuah program *executable*, dan import "fmt" yang menyertakan pustaka (library) untuk menangani *input* dan *output* (I/O).

Kita masuk ke func main(), fungsi utama tempat eksekusi dimulai. Di dalamnya, kita pertama-tama mendeklarasikan var a, b int untuk menyiapkan dua variabel bilangan bulat, a dan b.

Selanjutnya, fmt.Scan(&a, &b) dieksekusi. Program akan berhenti sejenak, menunggu pengguna memasukkan dua angka bulat yang dipisahkan spasi. Angka pertama akan disimpan ke dalam a, dan angka kedua akan disimpan ke dalam b. Dalam konteks program ini, a akan bertindak sebagai bilangan *basis* dan b sebagai bilangan *eksponen* atau *pangkat*.

Setelah mendapatkan masukan, baris result := 1 dieksekusi. Ini adalah langkah yang sangat krusial. Kita mendeklarasikan variabel baru bernama result (menggunakan sintaks deklarasi singkat :=) dan langsung menginisialisasinya dengan nilai 1. Kita tidak bisa memulainya dari 0, karena kita akan melakukan operasi perkalian; jika dimulai dari 0, hasil akhirnya akan selalu 0. Angka 1 adalah nilai *identitas* untuk operasi perkalian.

Bagian inti dari program ini adalah perulangan for i := 0; i < b; i++. Mari kita bedah:

1. **Inisialisasi:** i := 0. Sebuah variabel pencacah (iterator) i dibuat dan dimulai dari 0.
2. **Kondisi:** i < b. *Loop* ini akan terus berulang selama nilai i masih *kurang dari* nilai b (angka kedua yang dimasukkan pengguna).

3. **Increment:** i++. Setelah setiap putaran *loop* selesai, nilai i akan ditambah satu.

   Karena i dimulai dari 0 dan berhenti *sebelum* mencapai b, *loop* ini akan berputar tepat sebanyak b kali. (Misalnya, jika b=3, *loop* akan berjalan untuk i=0, i=1, dan i=2. Itu total 3 putaran).

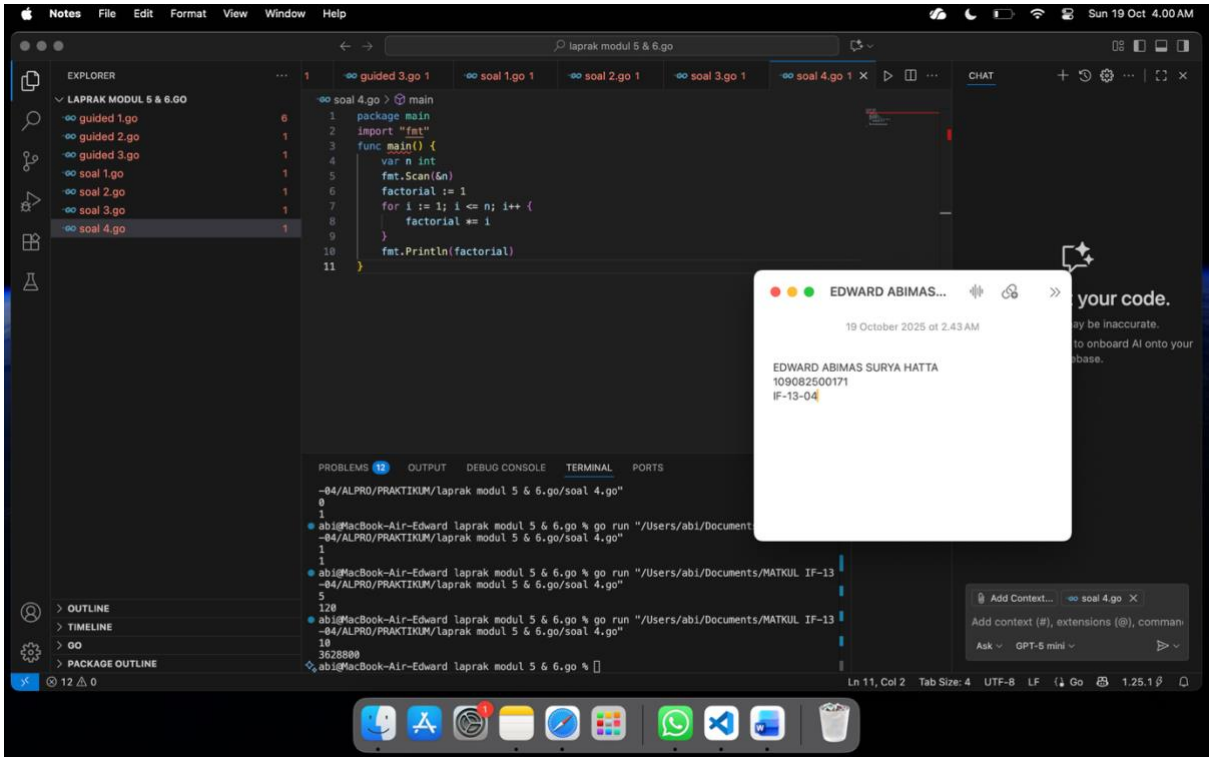
   Di dalam setiap putaran *loop*, perintah result *= a dieksekusi. Ini adalah kependekan dari result = result * a. Perintah ini mengambil nilai result saat ini, mengalikannya dengan nilai a, dan menyimpan hasil barunya kembali ke dalam result.

4. Tugas 3

Source code

```
package main
import "fmt"
func main() {
    var n int
    fmt.Scan(&n)
    factorial := 1
    for i := 1; i <= n; i++ {
        factorial *= i
    }
    fmt.Println(factorial)
}
```

Screenshoot program



Deskripsi program

Program ini, seperti biasa, diawali dengan package main (menandakan program *executable*) dan import "fmt" (untuk *input/output*).

Kita lalu masuk ke func main(), fungsi utama tempat program dieksekusi. Di dalamnya, kita pertama-tama mendeklarasikan var n int, sebuah variabel n bertipe integer. Variabel ini akan menampung angka yang ingin kita hitung faktorialnya.

Selanjutnya, fmt.Scan(&n) akan dieksekusi. Program akan berhenti di sini dan menunggu pengguna memasukkan satu angka bulat, yang kemudian akan disimpan ke dalam variabel n tersebut.

Baris berikutnya adalah factorial := 1. Ini adalah langkah deklarasi sekaligus inisialisasi yang sangat penting. Kita membuat variabel baru bernama factorial dan memberinya nilai awal 1. Nilai awal 1 ini krusial karena kita akan melakukan operasi perkalian berulang. Dalam perkalian, 1 adalah nilai *identitas* (angka apa pun dikali 1 adalah dirinya sendiri). Jika kita menginisialisasinya dengan 0, hasil akhirnya akan selalu 0.

Inti dari program ini adalah perulangan for i := 1; i <= n; i++. Mari kita bedah:

- **i := 1: Kita membuat variabel pencacah (iterator) i dan memulainya dari 1. Kita tidak mulai dari 0 karena mengalikan dengan 0 akan membuat hasilnya 0.**
- **i <= n: Ini adalah kondisi. Perulangan akan terus berlanjut selama nilai i masih kurang dari atau sama dengan n(angka yang dimasukkan pengguna).**
- **i++: Ini dieksekusi di akhir setiap putaran, menaikkan nilai i sebesar satu.**

Di dalam *loop*, kita hanya punya satu perintah: factorial *= i. Ini adalah sintaks singkat untuk factorial = factorial * i. Perintah ini mengambil nilai factorial saat ini, mengalikannya dengan nilai i saat ini, dan menyimpan hasilnya kembali ke variabel factorial.