

Pertemuan 12

PENGUJIAN PERANGKAT LUNAK

1. DASAR-DASAR PENGUJIAN PL

- Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah PL secara manual maupun otomatis untuk menguji apakah PL sudah memenuhi persyaratan atau belum, atau untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya.
- Pengujian bertujuan untuk mencari kesalahan.
- Pengujian yang baik adalah pengujian yang memiliki kemungkinan besar dalam menemukan kesalahan sebanyak mungkin dengan usaha sekecil mungkin.

A. Tujuan Pengujian

- a. Menilai apakah perangkat lunak yang dikembangkan telah memenuhi kebutuhan pemakai.
- b. Menilai apakah tahap pengembangan perangkat lunak telah sesuai dengan metodologi yang digunakan.
- c. Membuat dokumentasi hasil pengujian yang menginformasikan kesesuaian perangkat lunak yang diuji dengan spesifikasi yang telah ditentukan.

B. *Testability*

Testability adalah kemampuan PL untuk dapat diuji artinya seberapa mudah sebuah program komputer untuk bisa diuji.

Karakteristik *testability* PL:

- a. Kemampuan untuk bisa dioperasikan (*operability*)
- b. Kemampuan untuk bisa diobservasi (*observability*)
- c. Kemampuan untuk dapat dikontrol (*controllability*)
- d. Kemampuan untuk dapat disusun (*decomposability*)
- e. Kesederhanaan (*simplicity*)
- f. Stabilitas (*stability*)
- g. Kemampuan untuk dapat dipahami (*understandability*)

C. Karakteristik Pengujian

- a. Pengujian yang baik memiliki probabilitas tinggi untuk menemukan kesalahan
- b. Pengujian yang baik tidak berulang-ulang, waktu dan sumber daya pengujian terbatas
- c. Pengujian terbaik harus menjadi “bibit terbaik” yaitu pengujian yang memiliki kemungkinan tertinggi dalam mengungkap seluruh kelas kesalahan
- d. Pengujian yang baik tidak terlalu sederhana atau tidak terlalu rumit

2. PENGUJIAN WHITE BOX

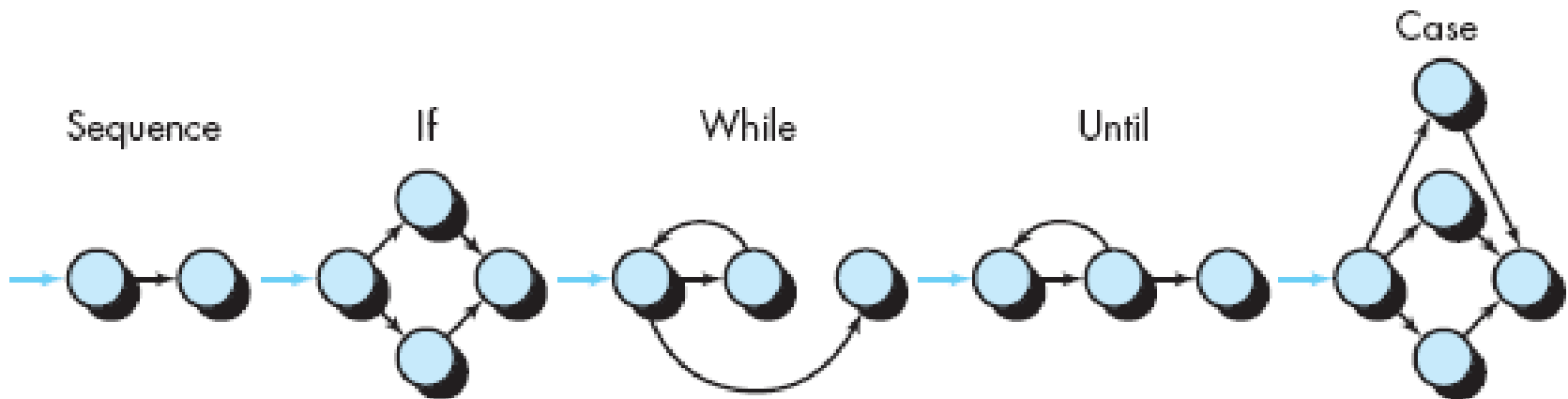
- Disebut juga pengujian kotak kaca (*glass box testing*).
- Merupakan sebuah filosofi perancangan *test case* yang menggunakan struktur kontrol.
- *Test case* pada *white box*:
 - a. Menjamin bahwa semua jalur independen di dalam modul telah dieksekusi sedikitnya satu kali
 - b. Melaksanakan semua keputusan logis pada sisi benar dan salah
 - c. Melaksanakan semua perulangan (*loop*) yang memenuhi semua batas operasional
 - d. Melakukan struktur data internal untuk memastikan kebenarannya

A. Pengujian Jalur Dasar (*Basis Path Testing*)

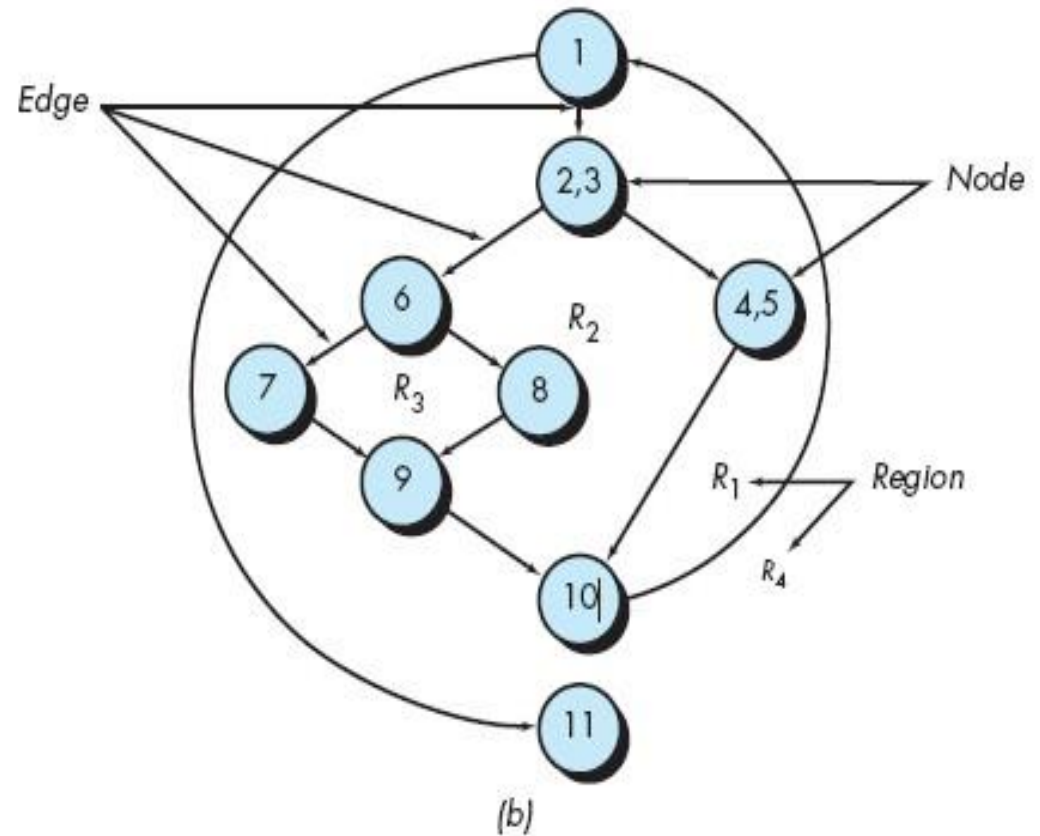
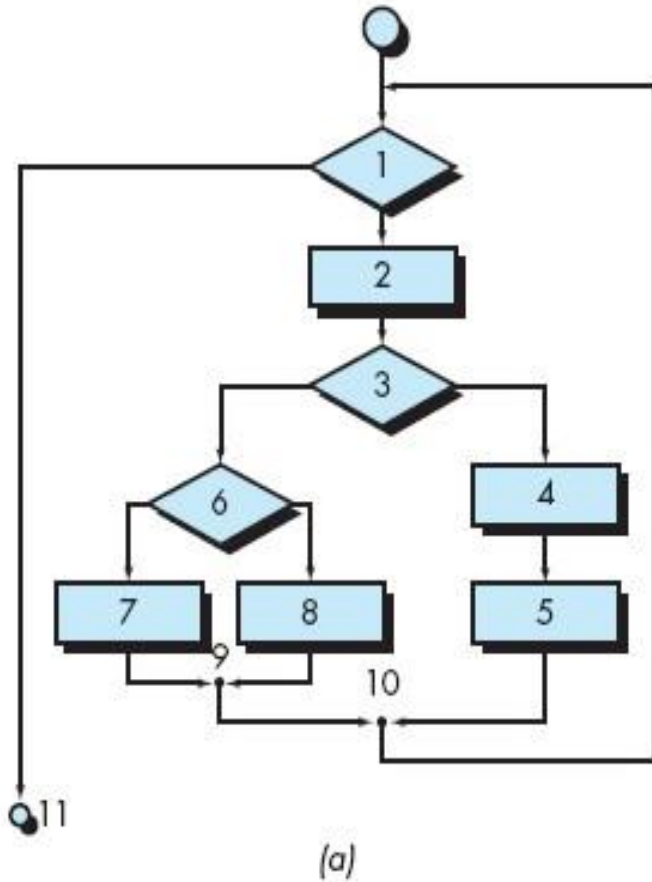
- Adalah teknik pengujian yang memungkinkan perancangan *test case* untuk menurunkan ukuran kompleksitas logis dari suatu rancangan prosedural dan menggunakan ukuran ini sebagai pedoman untuk menentukan rangkaian dasar jalur eksekusi.
- *Test case* diturunkan untuk menguji rangkaian dasar yang dijamin untuk mengeksekusi setiap pernyataan dalam program, setidaknya satu kali selama pengujian.
- Menggambarkan arus kontrol logis dengan menggunakan Notasi Grafik Alir (*Flow Graph*)

a. Notasi Grafik Alir (*Flow Graph*)

- Adalah notasi sederhana untuk merepresentasikan aliran kontrol logis.
- Lingkaran mewakili pernyataan kode program
- Notasi *flow graph* seperti gambar di bawah ini:



Untuk menggambarkan *flow graph*, dengan merepresen-
tasikan perancangan prosedural seperti gambar berikut:



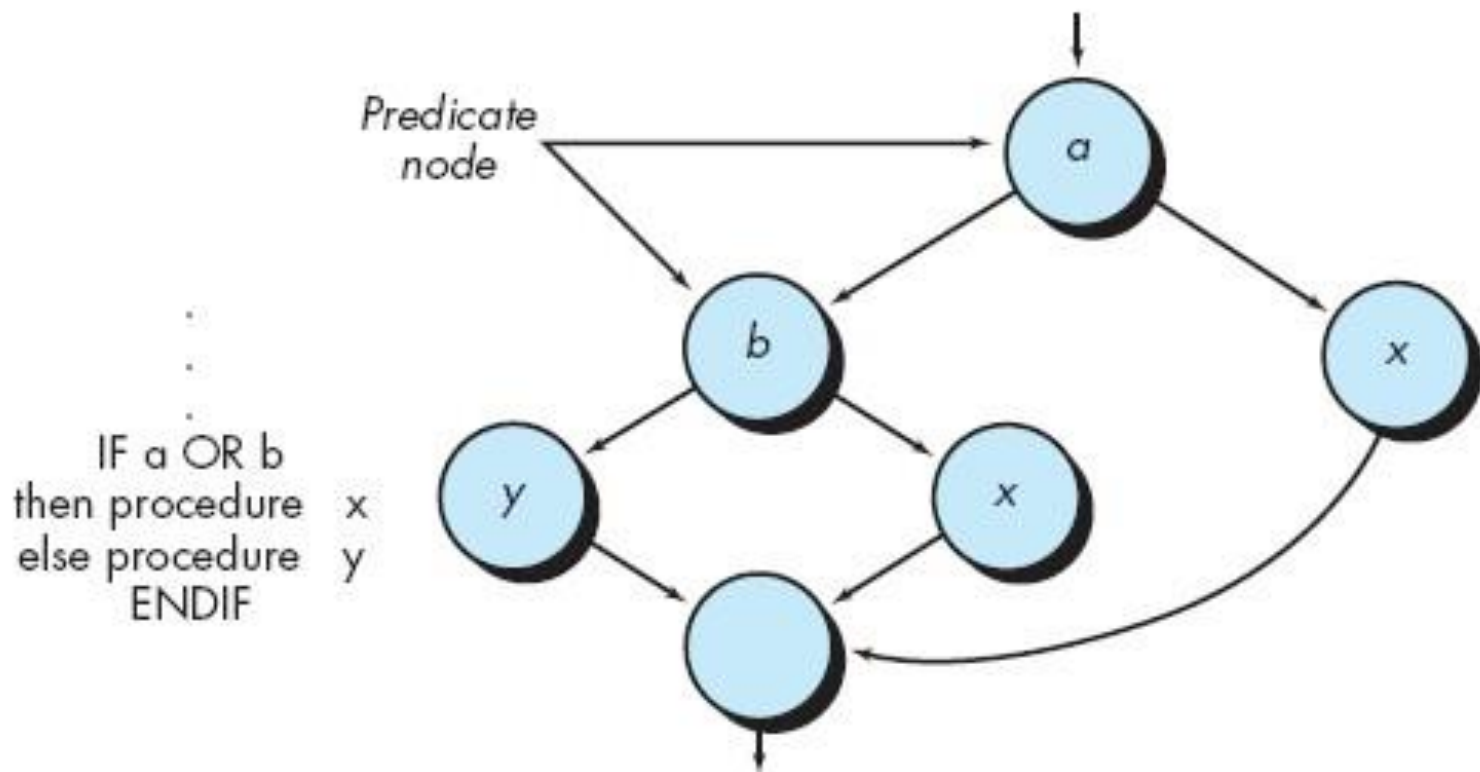
Notasi Flowchart

Flow Graph

b. Notasi *Flow Graph* (Lanjutan)

- Lingkaran menunjukkan **simpul** (***node***), merupakan satu atau lebih pernyataan-pernyataan prosedural
- Panah menunjukkan ***edge*** atau ***link***, merupakan aliran kendali
- Area yang dibatasi oleh *edge* dan *node* disebut ***region***.
- *Flow graph* menjadi rumit ketika adanya kondisi gabungan pada saat satu atau operator *boolean* ada dalam pernyataan bersyarat.
- *Node* yang berisi kondisi disebut ***node predikat*** dan ditandai oleh dua atau lebih *edge* yang berasal dari *node* tersebut.

Logika gabungan

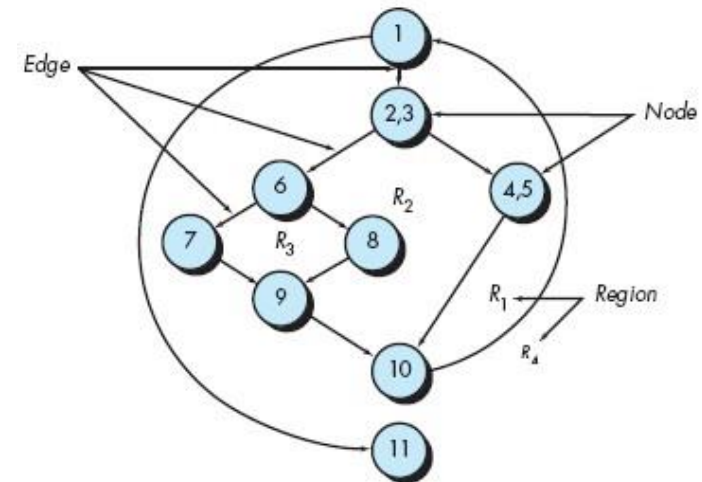


B. Jalur Independen

- Jalur Independen (*Independent Path*) adalah setiap jalur yang melalui program yang memperkenalkan setidaknya satu kumpulan pernyataan-pernyataan pemrosesan atau kondisi baru.

Jalur independen gambar disamping:

- Path 1:** 1-11
- Path 2:** 1-2-3-4-5-10-1-11
- Path 3:** 1-2-3-6-8-9-10-1-11
- Path 4:** 1-2-3-6-7-9-10-1-11



- Path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 tidak dianggap jalur independen karena tidak melintasi setiap *edge* baru
- Path 1 sampai 4 merupakan **basis set**.

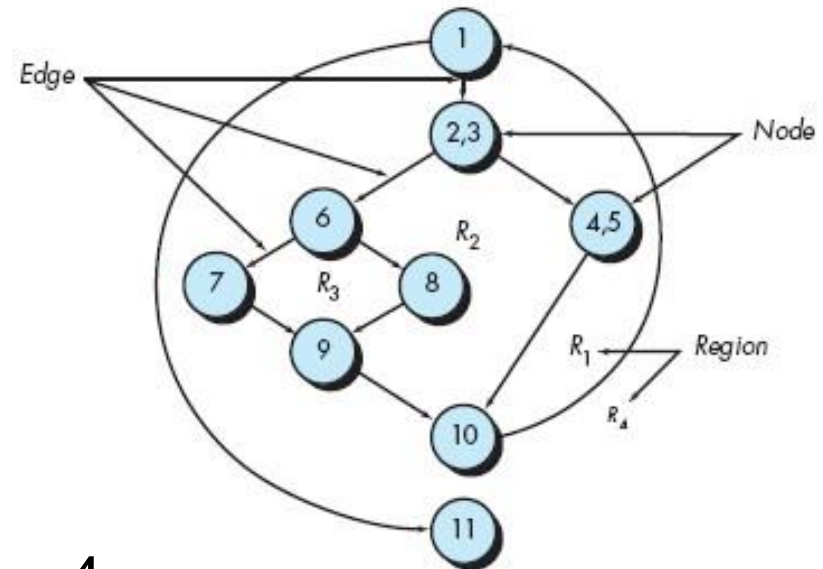
C. Kompleksitas Siklomatik

- Adalah metrik PL yang menyediakan ukuran kuantitatif dari kompleksitas logis suatu program.
- Perhitungan Kompleksitas Siklomatik:
 - a. Jumlah daerah-daerah (*region*) *flow graph* yang berhubungan dengan Kompleksitas Siklomatik
 - b. Kompleksitas Siklomatik $V(G) = E - N + 2$ dimana E adalah jumlah *edge*, N adalah jumlah *node*.
 - c. Kompleksitas Siklomatik $V(G) = P + 1$ dimana P adalah jumlah *node* predikat.

Kompleksitas Siklomatik (Lanjutan)

Dari kasus *independent path*:

- Jumlah *region* adalah 4
- $V(G) = 11 \text{ edge} - 9 \text{ node} + 2 = 4$
- $V(G) = 3 \text{ node predikat} + 1 = 4$
- Jadi Kompleksitas Siklomatiknya adalah 4



D. Menghasilkan *Test Case*

Diberikan Pseudocode sbb:

- 1 → do while record masih ada
 - baca record
- 2 → if record ke 1 = 0
- 3 → then proses record simpan di buffer
naikan counter
- 4 → • else if record ke 2 = 0
- 5 →
 - then reset counter
- 6 → else proses record
simpan pada file
- 7a → endif
- endif
- 7b → enddo
- 8 → end

Langkah-langkah untuk menurunkan *basis set*:

1. Buat *flow graph*
2. Menentukan *independent path*
3. Menentukan kompleksitas siklomatik

Catatan: Dosen menjelaskan langkah-langkah di atas

E. Pengujian Struktur Kontrol

a. Pengujian Kondisi

- Pengujian kondisi adalah metode perancangan *test case* yang menguji kondisi logis yang terdapat dalam modul program.
- Kondisi sederhana adalah variabel *boolean* atau ekspresi relasional, kemungkinan didahului oleh satu operator NOT
- Jenis kesalahan dalam kondisi meliputi kesalahan operator *boolean*, kesalahan variabel *boolean*, kesalahan kurung *boolean*, kesalahan operator relasional, dan kesalahan ekspresi aritmatika.

b. Pengujian Perulangan

Adalah teknik pengujian *white box* yang fokus pada validitas konstruksi perulangan.

(1) Perulangan Sederhana

Pengujian dilakukan dengan mudah, dimana n jumlah maksimum yang diijinkan melewati perulangan:

- (a) Melewati perulangan secara keseluruhan
- (b) Hanya satu kali melalui perulangan
- (c) Dua kali melalui perulangan
- (d) Melalui perulangan sebanyak m dimana $m < n$
- (e) $n - 1$, n , $n + 1$ melalui perulangan

Contoh Perulangan Sederhana

Berikut diberikan pseudocode tentang perulangan sederhana, dan carilah hasilnya.

x : integer

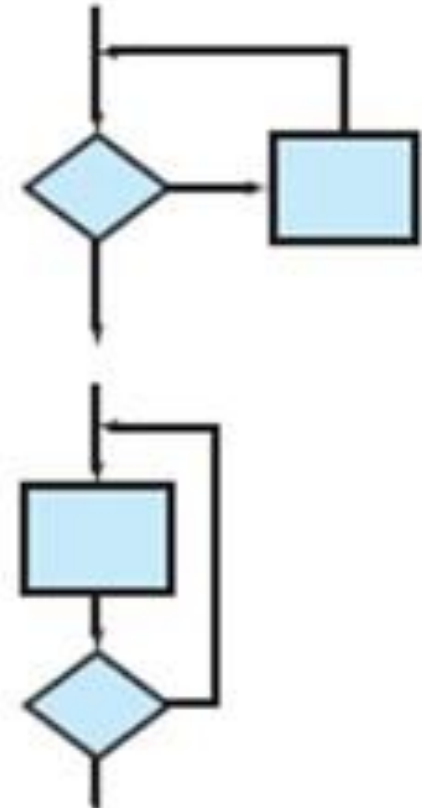
x = 1

WHILE (x < 5) DO

PRINT (x)

x = x + 1

ENDWHILE



(2). Perulangan Bersarang

Menggunakan pendekatan perulangan sederhana, sehingga jumlah pengujian akan meningkat.

Petunjuk pengujian:

- (a) Mulai dari perulangan terdalam dan atur semua perulangan ke nilai minimum
- (b) Lakukan pengujian perulangan sederhana untuk perulangan terdalam, sambil menjaga perulangan luar pada nilai minimum (misal *counter* perulangan)
- (c) Lanjutkan pada perulangan ke luar dan lakukan pengujian pada perulangan berikutnya
- (d) Lakukan sampai semua perulangan telah diuji

Contoh Perulangan Bersarang

Berikut diberikan pseudocode tentang perulangan bersarang, dan carilah hasilnya.

i, j : integer

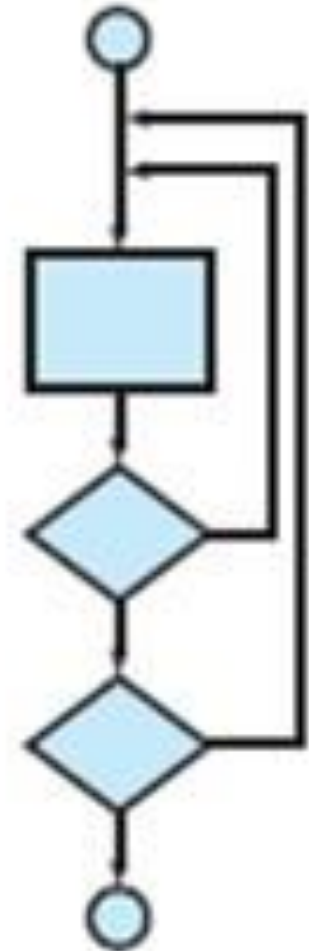
FOR $i = 1$ TO 4 DO

FOR $j = 1$ TO 3 DO

PRINT (i, j)

ENDFOR

ENDFOR



(3). Perulangan Terangkai

- Pengujian menggunakan pendekatan perulangan sederhana bila masing-masing perulangan independen.
- Tetapi bila dua perulangan dirangkai dan *counter* perulangan 1 digunakan sebagai harga awal perulangan 2 maka perulangan tersebut menjadi tidak independen, dan direkomendasikan ke perulangan tersarang

Contoh Perulangan Terangkai

x, y : integer

x = 0

y = 1

WHILE (x < 20) DO

PRINT (x)

x = x + 2

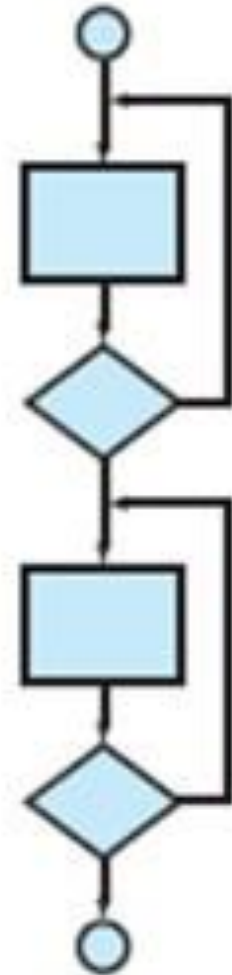
ENDWHILE

WHILE (y < 20) DO

PRINT (y)

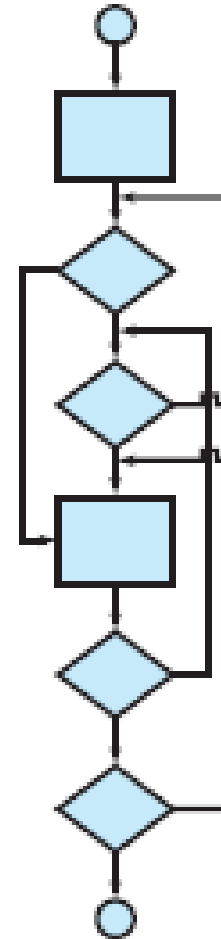
y = y + 2

ENDWHILE



(4). Perulangan Tak Terstruktur

Kapan saja memungkinkan, perulangan didisain kembali agar mencerminkan penggunaan konstruksi pemrograman terstruktur.



3. PENGUJIAN BLACK BOX

- Disebut juga pengujian perilaku.
- Pengujian *black box* memungkinkan untuk membuat beberapa kumpulan **kondisi input** yang akan melakukan semua kebutuhan fungsional untuk program.
- Kategori kesalahan pada pengujian *black box*:
 - a. Fungsi yang salah atau hilang
 - b. Kesalahan antarmuka
 - c. Kesalahan struktur data atau akses basis data eksternal
 - d. Kesalahan perilaku atau kinerja
 - e. Kesalahan inisialisasi dan penghentian

1. Metode Pengujian Berbasis Grafik

Langkah-langkah pengujian:

- Memahami objek-objek yang dimodelkan dalam PL dan penghubung yang menghubungkan objek-objek tersebut
- Menentukan serangkaian pengujian yang memastikan bahwa semua objek memiliki hubungan satu sama lain seperti yang diharapkan

Metode Pengujian Berbasis Grafik (Lanjutan)

- Node direpresentasikan sebagai lingkaran.
- Hubungan direpresentasikan dengan anak panah
- Hubungan satu arah (*directed link*) bahwa hubungan bergerak hanya satu arah.
- Hubungan dua arah atau hubungan simetris (*bidirection link*) bahwa hubungan berlaku dua arah.
- Hubungan paralel digunakan ketika ada sejumlah hubungan yang berbeda yang dibangun di antara *node-node* grafik.

2. Partisi Kesetaraan (*Equivalence Partitioning*)

- Adalah metode pengujian *black box* yang membagi daerah *input* program ke dalam kelas-kelas data dari *test case* yang dapat diturunkan.
- Sebuah kelas kesetaraan merepresentasikan keadaan valid atau tidak valid dari kondisi *input*.
- Contoh: kesalahan terhadap semua data karakter yang mungkin mengharuskan banyak *test case* sebelum kesalahan umum teramati.

Partisi Kesetaraan (Lanjutan)

Kelas kesetaraan dapat didefinisikan:

- Jika kondisi *input* menspesifikasikan range, satu kelas kesetaraan yang valid dan dua kelas kesetaraan yang tidak valid didefinisikan
- Jika kondisi *input* membutuhkan nilai tertentu, satu kelas kesetaraan yang valid dan dua kelas kesetaraan yang tidak valid didefinisikan
- Jika kondisi *input* menspesifikasikan anggota dari himpunan, satu kelas kesetaraan yang valid dan dua kelas kesetaraan yang tidak valid didefinisikan
- Jika kondisi *input* adalah *boolean*, satu kelas kesetaraan yang valid dan dua kelas kesetaraan yang tidak valid ditentukan

Contoh Pengujian *Equivalence Partitioning*

Ketentuan diskon 20% untuk pembelian barang sejenis sebanyak 2, dan diskon 30% untuk pembelian barang sejenis sebanyak 3 atau lebih

No	Test Case	Hasil yang diharapkan	Hasil yang diperoleh	Ket
1.	User mengisi jumlah barang = 0	Sistem tidak memproses transaksi dan menampilkan umpan balik	Sistem menampilkan pesan bahwa jumlah pembelian harus diisi	Berhasil
2.	User mengisi jumlah barang = 1	Sistem akan menghitung diskon dan subtotal	Sistem menghitung diskon sebesar 0 dan menghitung subtotal	Berhasil
3.	User mengisi jumlah barang = 2	Sistem akan menghitung diskon dan subtotal	Sistem menghitung diskon sebesar 20% dan menghitung subtotal	Berhasil
4.	User mengisi jumlah barang ≥ 3	Sistem akan menghitung diskon dan subtotal	Sistem menghitung diskon sebesar 30% dan menghitung subtotal	Berhasil

3. Analisis Nilai Batas (*Boundary Value Analysis*)

- Merupakan teknik perancangan *test case* yang melengkapi partisi kesetaraan dengan fokus pada kondisi *input*, dan juga akan menghasilkan *output*.
- Banyak kesalahan terjadi pada kesalahan *input*.
- BVA mengijinkan untuk menyeleksi kasus uji yang menguji batasan nilai input.
- BVA merupakan komplemen dari *equivalence partitioning*, lebih memilih pada elemen-elemen di dalam kelas ekivalen pada bagian sisi batas dari kelas

Pedoman BVA

- a. Jika kondisi *input* menspesifikasikan range yang dibatasi oleh nilai a dan b, *test case* harus dirancang dengan nilai a dan b dan hanya di atas dan di bawah nilai a dan b
- b. Jika kondisi *input* menspesifikasikan sejumlah nilai, *test case* harus dikembangkan untuk menguji jumlah-jumlah minimum dan maksimum.
- c. Terapkan pedoman 1 dan 2 untuk kondisi *input*.
- d. Jika struktur data program internal memiliki batas-batas yang telah ditentukan, pastikan untuk merancang *test case* untuk menguji struktur data pada batasnya.

Contoh Pengujian BVA

Jika sistem menampilkan jumlah stok barang adalah n , maka:

Noc	Test Case	Hasil yang diharapkan	Hasil yang diperoleh	Ket
1.	User mengisi stok = 0	Sistem tidak memproses transaksi dan memberikan umpan balik	Sistem menampilkan pesan bahwa jumlah pembelian harus diisi	Berhasil
2.	User mengisi stok = 1 sampai n	Sistem akan memproses transaksi dan mengurangi jumlah stok	Sistem menghitung subtotal dan mengurangi jumlah stok	Berhasil
3.	User mengisi stok $> n$	Sistem tidak memproses transaksi dan memberikan umpan balik	Sistem menampilkan pesan bahwa jumlah pembelian melebihi stok	Berhasil

Catatan: sebaiknya tidak menuliskan “sesuai harapan” pada kolom ini

4. Pengujian Larik Ortogonal

- Dapat diterapkan untuk masalah-masalah dimana *input domain* relatif kecil tapi terlalu besar untuk mengakomodasi pengujian yang lengkap.
- Bermanfaat dalam menemukan kesalahan yang terkait dengan logika yang salah dalam komponen PL

Contoh:



LOGO

Email

Password

LOGIN CANCEL

Jika rancangan aplikasi untuk login seperti gambar di atas, maka *black box testing* dapat dibuat berdasarkan beberapa kondisi input (5 kondisi)

Contoh pengujian keamanan pada User Login

No	Test Case	Hasil yang diharapkan	Hasil yang diperoleh	Ket
1.	User tidak mengisi <i>username</i> atau <i>password</i> atau keduanya	User tidak bisa masuk ke dalam sistem	Sistem tetap pada form Login dan menampilkan pesan untuk mengisi <i>username</i> dan <i>password</i>	Berhasil
2.	User mengisi <i>username</i> saja atau <i>password</i> saja	User tidak bisa masuk ke dalam sistem	Sistem akan menampilkan pesan untuk mengisi <i>username</i> atau <i>password</i>	Berhasil
3.	User salah mengisi <i>username</i>	User tidak bisa masuk ke dalam sistem	Sistem akan menampilkan pesan jika <i>username</i> atau <i>password</i> salah	Berhasil
4.	User salah mengisi <i>password</i>	User tidak bisa masuk ke dalam sistem	Sistem akan menampilkan pesan jika <i>username</i> atau <i>password</i> salah	Berhasil
5.	User salah mengisi <i>username</i> dan <i>password</i>	User tidak bisa masuk ke dalam sistem	Sistem akan menampilkan pesan jika <i>username</i> atau <i>password</i> salah	Berhasil

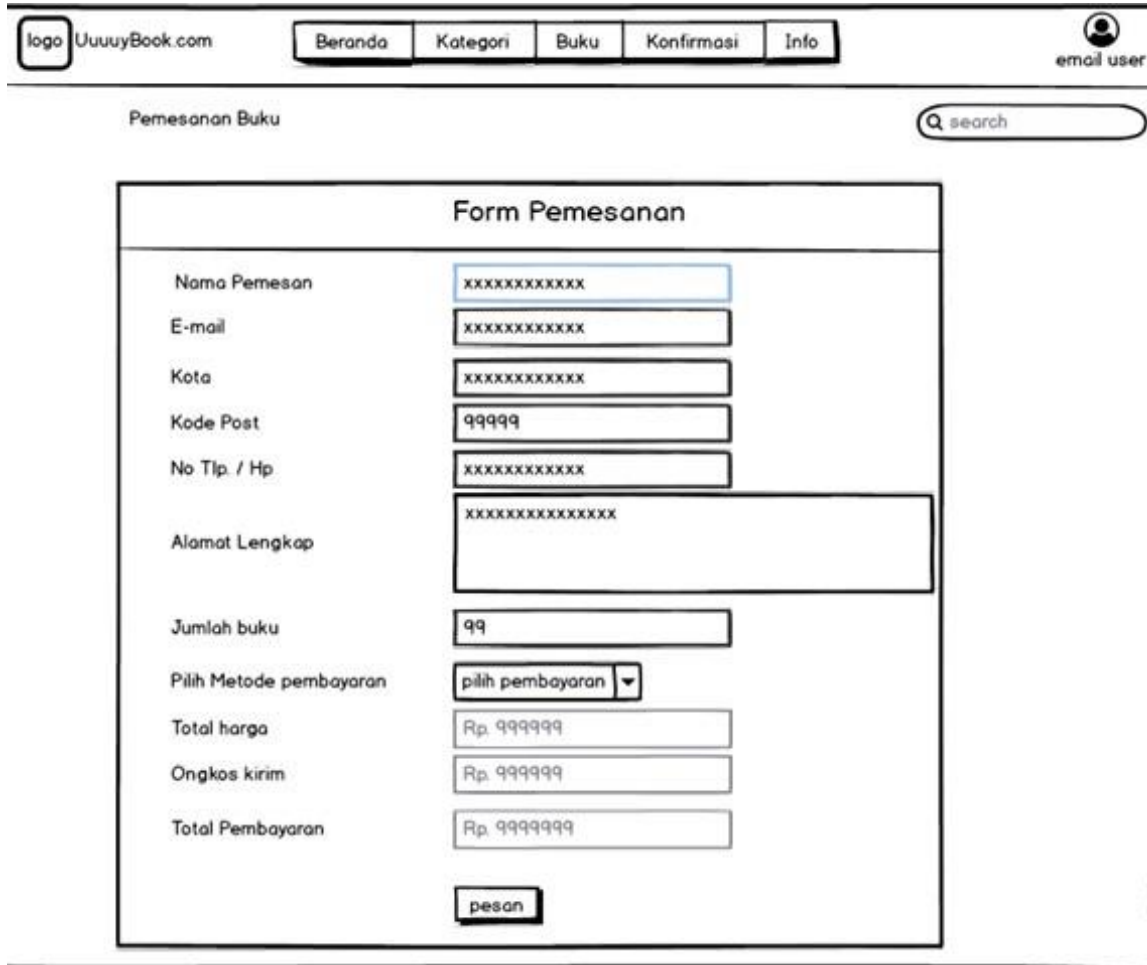
Latihan 1

Buatlah Flow graph kemudian tentukan jalur independen, dan hitung Kompleksitas Siklomatik

```
00: int bsearch(int x, const int A[], int N) {  
01:     int begin = 0;  
02:     int end = N;  
03:     while (begin < end) {  
04:         int middle = begin + (end - begin) / 2;  
05:         if (x <= A[middle])  
06:             end = middle;  
07:         else  
08:             begin = middle + 1;  
09:     }  
10:     return begin;  
11: }
```

Latihan 2

Buat deskripsi black box testing jika diberikan gambar berikut



UuuuyBook.com Beranda Kategori Buku Konfirmasi Info email user

Pemesanan Buku search

Form Pemesanan

Nama Pemesan	xxxxxxxxxxxx
E-mail	xxxxxxxxxxxx
Kota	xxxxxxxxxxxx
Kode Post	99999
No Tlp. / Hp	xxxxxxxxxxxx
Alamat Lengkap	xxxxxxxxxxxxxxxx
Jumlah buku	99
Pilih Metode pembayaran	pilih pembayaran ▼
Total harga	Rp. 999999
Ongkos kirim	Rp. 999999
Total Pembayaran	Rp. 9999999

pesan

TUGAS

PROCEDURE RATA-RATA

- INTERFACE RESULT rata, total, input, total.valid
- INTERFACE RESULT nilai, minim, max
- TYPE NILAI (1:100) IS SCALAR ARRAY;
- TYPE rata, total. input, total.valid, max.minim, jumlah IS SCALAR;
- TYPE I IS INTEGER;
- I = 1;
- total. input = total. valid = 0;
- jumlah = 0;
- DO WHILE nilai(i) <> -999 .and. total.input < 100
 - tambahkan total.input dengan 1;
 - IF nilai(i) >= minimum .and. nilai(i) <=max;
 - THEN tambahkan total.valid dengan I;
 - jumlah=jumlah + nilai(i);
 - ELSE skip;
 - END IF
 - tambahkan i dengan 1;
- ENDDO
- IF total. valid > 0
- THEN rata =jumlah/total. valid;
- ELSE rata = -999;
- ENDIF
- END

TUGAS

Dari pseudocode di atas, buatlah

1. *Flow graph*
2. Tentukan *independent path*
3. Tentukan kompleksitas siklomatik