**LAPORAN PRAKTIKUM  
PENGUJIAN PERANGKAT LUNAK**

**HALAMAN JUDUL**

****

**Disusun Oleh :**

**NAMA : MUHAMMAD FATIH ‘AD-LI**

**NIM : 32602200017**

**PRODI TEKNIK INFORMATIKA**

**FAKULTAS TEKNOLOGI INDUSTRI**

**UNIVERSITAS ISLAM SULTAN AGUNG**

**SEMARANG**

**2025**

****

****

# HALAMAN PENGESAHAN

****

**UNIVERSITAS ISLAM SULTAN AGUNG**

# 

# KATA PENGANTAR

Puji syukur penulis panjatkan kepada ALLAH SWT, yang telah memberikan rahmat, taufik serta hidayah-Nya, sehingga laporan Praktikum Pengujian Perangkat Lunak.

Tanpa lupa penulis mengucapkan terima kasih kepada :

1. Rektor UNISSULA Bapak Prof. Dr. H. Gunarto, S.H, M.Hum yang mengijinkan penulis menimba ilmu di kampus ini
2. Dekan Fakultas Teknologi Industri Ibu Dr. Ir. Hj, Novi Marlyana, S. T.,

M. T., IPU., ASEAN Eng.

1. Dosen pengampu penulis Bapak Andi Riansyah, ST., M. Kom yang telah memberi ilmu tentang Pengujian Perangkat Lunak
2. Asisten dosen yang telah membantu menjelasan menegenai Praktikum Pengujian Perangkat Lunak.
3. Orang tua penulis yang telah mengijinkan untuk menyelesaikan laporan ini,
4. Dan kepada semua pihak yang tidak dapat saya satu persatu.

Penulis menyadari bahwa dalam penyusunan laporan ini masih terdapat banyak kekurangan, untuk itu penulis mengharap kritik dan saran dari pembaca untuk sempurnanya laporan ini. Semoga dengan ditulisnya laporan ini dapat menjadi sumber ilmu bagi setiap pembaca.

Semarang, 11 Januari 2025

Muhammad Fatih ‘Ad-Li

****

****

# 

# DAFTAR ISI

Table of Contents

[HALAMAN JUDUL i](#_Toc156397498)

[HALAMAN PENGESAHAN ii](#_Toc156397499)

[KATA PENGANTAR iii](#_Toc156397500)

[DAFTAR ISI iv](#_Toc156397501)

[DAFTAR GAMBAR vii](#_Toc156397502)

[DAFTAR TABEL ix](#_Toc156397503)

[BAB I PENGUJIAN *WHITEBOX* (1): PENGUJIAN *BASIS PATH* 1](#_Toc156397504)

[1.1 Tujuan 1](#_Toc156397505)

[1.2 Alat dan Bahan 1](#_Toc156397506)

[1.3 Dasar Teori 1](#_Toc156397507)

[1.3.1 Teknik Pengujian Perangkat Lunak 1](#_Toc156397508)

[1.3.2 Tujuan Pengujian 1](#_Toc156397509)

[1.3.3 Prinsip Pengujian 2](#_Toc156397510)

[1.3.4 Perancangan Kasus Uji/Test Case 2](#_Toc156397511)

[1.3.5 Pengujian White Box/Glass Box 2](#_Toc156397512)

[1.3.6 Basis Path Testing 2](#_Toc156397513)

[1.4 Prosedur Praktikum 4](#_Toc156397514)

[1.5 Hasil Praktikum 6](#_Toc156397515)

[1.6 Analisa 8](#_Toc156397516)

[1.7 Kesimpulan 11](#_Toc156397517)

[BAB II PENGUJIAN *WHITEBOX* (2): PENGUJIAN KONDISI 12](#_Toc156397518)

[2.1 Tujuan 12](#_Toc156397519)

[2.2 Alat dan Bahan 12](#_Toc156397520)

[2.3 Dasar Teori 12](#_Toc156397521)

[2.3.1 Teknik Pengujian Perangkat Lunak 12](#_Toc156397522)

[2.4 Prosedur Praktikum 14](#_Toc156397523)

[2.5 Hasil Praktikum 17](#_Toc156397524)

[2.5.1 Pengujian Sesuai Tabel Keputusan 17](#_Toc156397525)

[2.6 Analisa 22](#_Toc156397526)

[2.7 Kesimpulan 26](#_Toc156397527)

[BAB III PENGUJIAN *BLACKBOX* (1): PENGUJIAN ANALISIS KELAS EKUIVALENSI 27](#_Toc156397528)

[3.1 Tujuan 27](#_Toc156397529)

[3.2 Alat dan Bahan 27](#_Toc156397530)

[3.3 Dasar Teori 27](#_Toc156397531)

[3.3.1 Pengujian *Black Box* 27](#_Toc156397532)

[3.3.2 Analisis Partisi Kelas Ekuivslensi (*Equivalence Class Partitioning*) 28](#_Toc156397533)

[3.4 Prosedur Praktikum 29](#_Toc156397534)

[3.5 Hasil Praktikum 36](#_Toc156397535)

[3.5.1 Pengujian Sesuai Tabel Keputusan 36](#_Toc156397536)

[3.6 Analisa 42](#_Toc156397537)

[3.7 Kesimpulan 46](#_Toc156397538)

[BAB IV PENGUJIAN *BLACKBOX* (2): PENGUJIAN ANALISIS NILAI BATAS 48](#_Toc156397539)

[4.1 Tujuan 48](#_Toc156397540)

[4.2 Alat dan Bahan 48](#_Toc156397541)

[4.3 Dasar Teori 48](#_Toc156397542)

[4.3.1 Analisis Nilai Batas (*Boundary Value Analysis* atau BVA) 48](#_Toc156397543)

[4.3.2 Petunjuk Pengujian *Boundary Value Analysis* (BVA) 48](#_Toc156397544)

[4.4 Prosedur Praktikum 49](#_Toc156397545)

[4.5 Hasil Praktikum 53](#_Toc156397546)

[4.5.1 Pengujian Sesuai Tabel Keputusan 53](#_Toc156397547)

[4.6 Analisa 56](#_Toc156397548)

[4.7 Kesimpulan 58](#_Toc156397549)

[BAB V PENGUJIAN FUNGSIONAL DENGAN SELENIUM IDE 59](#_Toc156397550)

[5.1 Tujuan 59](#_Toc156397551)

[5.2 Alat dan Bahan 59](#_Toc156397552)

[5.3 Dasar Teori 59](#_Toc156397553)

[5.3.1 Pengenalan *Selenium* 59](#_Toc156397554)

[5.3.2 Selenium IDE 60](#_Toc156397555)

[5.3.3 Mengeset *Recording* dan *Run* 63](#_Toc156397556)

[5.3.4 Menjalankan *Test Case* 63](#_Toc156397557)

[5.3.5 *Test Suite* 64](#_Toc156397558)

[5.4 Prosedur Praktikum 64](#_Toc156397559)

[5.5 Hasil Praktikum 69](#_Toc156397560)

[5.6 Analisa 70](#_Toc156397561)

[5.7 Kesimpulan 72](#_Toc156397562)

[BAB VI PENGUJIAN *STRESS* DENGAN *APACHE JMETER* 73](#_Toc156397563)

[6.1 Tujuan 73](#_Toc156397564)

[6.2 Dasar Teori 73](#_Toc156397565)

[6.3 Alat dan Bahan 75](#_Toc156397566)

[6.4 Prosedur Praktikum 75](#_Toc156397567)

[6.5 Hasil Praktikum 81](#_Toc156397568)

[6.6 Analisa 84](#_Toc156397569)

[6.7 Kesimpulan 85](#_Toc156397570)

[DAFTAR PUSTAKA 87](#_Toc156397571)

[LAMPIRAN 88](#_Toc156397572)

****

**UNIVERSITAS ISLAM SULTAN AGUNG**

# DAFTAR GAMBAR

[Gambar 1. 1 Notasi Diagram Alir 3](#_Toc156394800)

[Gambar 1. 2 Diagram Alir 3](#_Toc156394801)

[Gambar 1. 3 Grafik Alir 4](#_Toc156394802)

[Gambar 1. 4 Grafik Program 6](#_Toc156394803)

[Gambar 1. 5 Tabel Pre-Kondisi 6](#_Toc156394804)

[Gambar 1. 6 Hasil Uji 1 7](#_Toc156394805)

[Gambar 1. 7 Hasil Uji 2 8](#_Toc156394806)

[Gambar 1. 8 Hasil Uji 3 8](#_Toc156394807)

[Gambar 1. 9 Hasil Uji 4 8](#_Toc156394808)

[Gambar 1. 10 Hasil Uji 5 8](#_Toc156394809)

[Gambar 2. 1 Output Hasil Pengujian Rules 1 17](#_Toc156394848)

[Gambar 2. 2 Output Hasil Pengujian Rules 2 17](#_Toc156394849)

[Gambar 2. 3 Output Hasil Pengujian Rules 3 18](#_Toc156394850)

[Gambar 2. 4 Output Hasil Pengujian Rules 4 18](#_Toc156394851)

[Gambar 2. 5 Output Hasil Pengujian Rules 7 18](#_Toc156394852)

[Gambar 2. 6 Output Hasil Pengujian Rules 9 19](#_Toc156394853)

[Gambar 2. 7 Output Hasil Pengujian Rules 10 19](#_Toc156394854)

[Gambar 2. 8 Output Hasil Pengujian Rules 11 19](#_Toc156394855)

[Gambar 3. 1 Output Test Case 1 Black Box Equivalence Class 36](#_Toc156394916)

[Gambar 3. 2 Output Test Case 2 Black Box Equivalence Class 37](#_Toc156394917)

[Gambar 3. 3 Output Test Case 3 Black Box Equivalence Class 37](#_Toc156394918)

[Gambar 3. 4 Output Test Case 4 Black Box Equivalence Class 37](#_Toc156394919)

[Gambar 3. 5 Output Test Case 5 Black Box Equivalence Class 38](#_Toc156394920)

[Gambar 3. 6 Output Test Case 6 Black Box Equivalence Class 38](#_Toc156394921)

[Gambar 3. 7 Output Test Case 7 Black Box Equivalence Class 38](#_Toc156394922)

[Gambar 3. 8 Output Test Case 8 Black Box Equivalence Class 39](#_Toc156394923)

[Gambar 3. 9 Output Test Case 9 Black Box Equivalence Class 39](#_Toc156394924)

[Gambar 3. 10 Output Test Case 10 Black Box Equivalence Class 39](#_Toc156394925)

[Gambar 3. 11 Output Test Case 11 Black Box Equivalence Class 40](#_Toc156394926)

[Gambar 3. 12 Output Test Case 12 Black Box Equivalence Class 40](#_Toc156394927)

[Gambar 3. 13 Output Test Case 13 Black Box Equivalence Class 40](#_Toc156394928)

[Gambar 3. 14 Output Test Case 14 Black Box Equivalence Class 41](#_Toc156394929)

[Gambar 3. 15 Output Test Case 15 Black Box Equivalence Class 41](#_Toc156394930)

[Gambar 3. 16 Output Test Case 16 Black Box Equivalence Class 41](#_Toc156394931)

[Gambar 3. 17 Output Test Case 17 Black Box Equivalence Class 42](#_Toc156394932)

[Gambar 4. 1 Output Test Case 1 Black Box Boundary Analysis Value 53](#_Toc156394950)

[Gambar 4. 2 Output Test Case 2 Black Box Boundary Analysis Value 53](#_Toc156394951)

[Gambar 4. 3 Output Test Case 3 Black Box Boundary Analysis Value 53](#_Toc156394952)

[Gambar 4. 4 Output Test Case 4 Black Box Boundary Analysis Value 54](#_Toc156394953)

[Gambar 4. 5 Output Test Case 5 Black Box Boundary Analysis Value 54](#_Toc156394954)

[Gambar 4. 6 Output Test Case 6 Black Box Boundary Analysis Value 54](#_Toc156394955)

[Gambar 4. 7 Output Test Case 7 Black Box Boundary Analysis Value 55](#_Toc156394956)

[Gambar 4. 8 Output Test Case 8 Black Box Boundary Analysis Value 55](#_Toc156394957)

[Gambar 5. 1 Panel Selenium IDE 61](#_Toc156394982)

[Gambar 5. 2 Website Selenium IDE 64](#_Toc156394983)

[Gambar 5. 3 Pop up "add extension" 65](#_Toc156394984)

[Gambar 5. 4 Berhasil menambahkan extension Selenium IDE pada Chrome 65](#_Toc156394985)

[Gambar 5. 5 Tab "extensions" 66](#_Toc156394986)

[Gambar 5. 6 Halaman awal Selenium IDE 66](#_Toc156394987)

[Gambar 5. 7 Hasil Test Case "Energy Efficient - Google Search" 69](#_Toc156394988)

[Gambar 5. 8 Hasil Test Case "Selenium RC - Google Search" 69](#_Toc156394989)

[Gambar 5. 9 Hasil Test Case "Energy Efficient - Penelusuran Google" 70](#_Toc156394990)

[Gambar 5. 10 Hasil Test Case "Energy Efficient - Yahoo Search" 70](#_Toc156394991)

[Gambar 6. 1 Pengecekan Java Version dengan CMD 75](#_Toc156395092)

[Gambar 6. 2 Pilih Pengaturan Bahasa 76](#_Toc156395093)

[Gambar 6. 3 Memilih File Direktori Instalasi Apache JMeter 77](#_Toc156395094)

[Gambar 6. 4 Start Menu Apache JMeter 77](#_Toc156395095)

[Gambar 6. 5 Menambahkan Shortcut Apache JMeter 78](#_Toc156395096)

[Gambar 6. 6 Informasi Terkait Penginstalan 78](#_Toc156395097)

[Gambar 6. 7 Proses Instalasi yang Sedang Berjalan 78](#_Toc156395098)

[Gambar 6. 8 Instalasi Apache JMeter yang Telah Selesai 79](#_Toc156395099)

[Gambar 6. 9 Tampilan Awal Aplikasi JMeter 79](#_Toc156395100)

[Gambar 6. 10 Menambahkan Thread Group Pada TestPlan 81](#_Toc156395101)

[Gambar 6. 11 Membuat HTTP Request Default 82](#_Toc156395102)

[Gambar 6. 12 Penambahan HTTP Request 82](#_Toc156395103)

[Gambar 6. 13 Menambahkan Listener Graph Result 83](#_Toc156395104)

[Gambar 6. 14 Menambahkan Listener View Result in Table 83](#_Toc156395105)

[Gambar 6. 15 Hasil Dalam Bentuk Grafik 84](#_Toc156395106)

[Gambar 6. 16 Hasil Dalam Bentuk Tabel 84](#_Toc156395107)

****

**UNIVERSITAS ISLAM SULTAN AGUNG**

# 

# DAFTAR TABEL

[Table 1. 1 Tabel Hasil Praktikum 6](#_Toc156383999)

[Table 2. 1 Tabel Keputusan atau Kondisi 15](#_Toc156395525)

[Table 2. 2 Hasil Pengujian Kondisi 21](#_Toc156395526)

[Table 3. 1 Weak Equivalence Class Testing 31](#_Toc156395904)

[Table 3. 2 Weak Robust Equivalence Class Testing 31](#_Toc156395905)

[Table 3. 3 Strong Robust Equivalence Class Testing 33](#_Toc156395906)

[Table 4. 1 Test Case Boundary Value Analysis 51](#_Toc156396136)

[Table 6. 1 Deskripsi Apache JMeter 74](#_Toc156396139)

**BAB I   
PENGUJIAN *WHITEBOX* (1): PENGUJIAN *BASIS PATH***

****

****

# 

# BAB I PENGUJIAN *WHITEBOX* (1): PENGUJIAN *BASIS PATH*

## Tujuan

Tujuan dari praktikum ini, adalah :

1. Praktikan dapat mempersiapkan tahapan pengujian.
2. Praktikan dapat merancag kasus uji.
3. Praktikan dapat melakukan pengujian perangkat lunak dengan teks *Basis Path.*

## Alat dan Bahan

1. C Compiler

## Dasar Teori

### **Teknik Pengujian Perangkat Lunak**

Pengujian *software* adalah elemen kritis dari jaminan kualitas *software* dan merupakan *review* akhir dari spesifikasi, perancangan dan pengkodean. Pada tahap awal dari pengembangan *software*, *engineer* berusaha untuk membangun *software* dari sebuah konsep abstrak menjadi implementasi nyata. Pada saat pengujian, *engineer* membuat serangkaian kasus uji yang bertujuan untuk “merusak” *software* yang telah dibuat.

### **Tujuan Pengujian**

Tujuan pengujian *software* menurut Glen Meyer sbb:

1. Pengujian adalah proses eksekusi sebuah program untuk menemukan “*error*”
2. Kasus uji yang baik adalah sesuatu yang bisa mengungkapkan kemungkinan yang tinggi untuk menemukan *error-error* yang tidak ditemukan sebelumnya
3. Pengujian yang sukses adalah sesuatu yang bisa mengungkapkan *error* tidak ditemukan sebelumnya.

### 

### **Prinsip Pengujian**

Pada saat melakukan pengujian terhadap suatu *software* ada beberapa prinsip pengujian yang harus diperhatikan, diantaranya :

1. Semua pengujian harus terlacak kebutuhan *user*
2. Pengujian harus direncanakan jauh sebelum pengujian dimulai
3. Pengujian dimulai dari kecil dan mengarah kepada yang besar
4. Pengujian yang sempurna adalah tidak mungkin
5. Agar pengujian berjalan efektif, maka sebaiknya dilakukan oleh pihak ketiga yang netral

### **Perancangan Kasus Uji/Test Case**

Setiap produk rekayasa perangkat lunak bisa diuji dalam dua cara yaitu

1. Mengetahui fungsinya, sehingga pengujian dilakukan dengan mendemonstasikan fungsi tersebut bisa berjalan dengan sempurna atau ada *error* (*Black box Testing*)
2. Mengetahui cara kerja internal dari produk tersebut (*White box Testing*)

### **Pengujian White Box/Glass Box**

Adalah sebuah metoda perancangan kasus uji yang menggunakan struktur kontrol dari perancangan prosedur. Pengguna metoda pengujian akan membuat *software engginer* dapat :

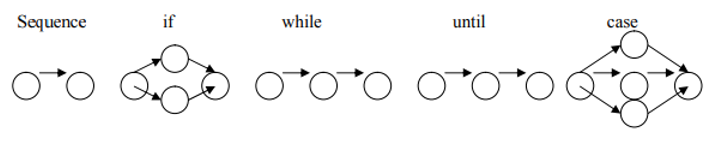
1. Menjamin bahwa semua jalur independen dalam sebuah modul telah dilewati paling tidak satu kali
2. Memeriksa semua keputusan logika baik pada sisi sebenarnya maupun pada sisi salahnya
3. Mengeksekusi semua *loop* pada nilai batasnya dan dalam nilai dimana dia harus

### **Basis Path Testing**

Metoda *Basis path testing* membuat perancang kasus uji bisa menurunkan sebuah ukuran kompleksitas logika dari sebuah perancangan prosedural dan ukuran ini selanjutnya digunakan untuk menentukan sekelompok data dasar (*basis set*) dari jalur eksekusi.

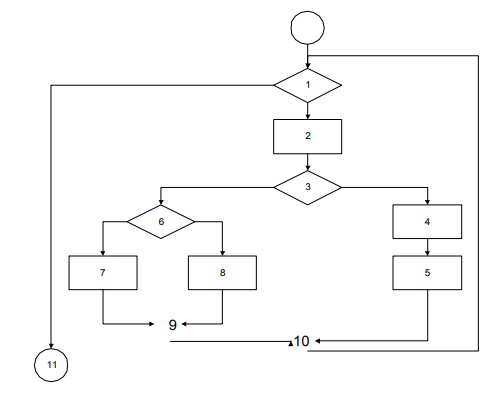
Pengujian *basis path* adalah teknik pengujian *white box* yg diusulkan Tom McCabe. Metode ini memungkinkan perancang *test case* mendapatkan ukuran kekompleksan *logical* dari perancangan prosedural dan menggunkan ukuran ini sebagai petunjuk untuk mendefinisikan *basis set* dari jalur pengerjaan. *Test case* yg didapat digunakan untuk mengerjakan *basis set* yg menjamin pengerjaan setiap perintah minimal satu kali selama pengujian.

1. **Notasi Diagram Alir**

****

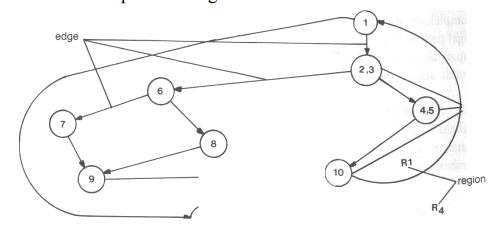
Gambar 1. 1 Notasi Diagram Alir

Untuk menggambarkan pemakaian diagram alir diberikan contoh perancangan prosedural dalam bentuk *flowchart*



Gambar 1. 2 Diagram Alir

Selanjutnya diagram alir diatas dipetakan ke grafik alir



Gambar 1. 3 Grafik Alir

Lingkaran/*node* :

menggambarkan satu/lebih perintah prosedural. Urutan proses dan keputusan dapat dipetakan dalam satu *node*.

Tanda panah/*edge* :

menggambarkan aliran kontrol. Setiap *node* harus mempunyai tujuan *node*

*Region* :

adalah daerah yg dibatasi oleh *edge* dan *node*. Termasuk daerah diluar grafik alir.

## Prosedur Praktikum

Berikut ini merupakan prosedur praktikum program pencarian biner dalam bahasa C.

* 1. Tentukan *Basis Path*
  2. Turunkan *Test Case*-nya
  3. Jalankan *Test Case* tersebut
  4. Analisis hasil pengujian tersebut!

**Program Pencarian Biner**

#include<stdio.h>

#include <stdlib.h>

int binsrc(int x[], int low, int high, int key)

{

int *mid*; 1

while(low <= high) 2

{

*mid* = (low + high)/2; if(x[*mid*] == key) 3

return *mid*; 8

if(x[*mid*] < key) 4

low = *mid*+1; 5

else

high=*mid*-1; 6

} 7

return -1; 8

} 9

int main()

{

printf("Nama\t: Muhammad Fatih ‘Ad-Li\n");

printf("NIM\t: 32602200017\n\n");

int a[20],key,i,n,succ; printf("Enter the n value\n"); scanf("%d",&n);

if(n>0)

{

printf("enter the elements in ascending order\n");

for(i=0; i<n; i++) scanf("%d",&a[i]);

printf("enter the key element to be searched\n");

scanf("%d",&key);

succ = binsrc(a,0,n-1,key); if(succ>=0)

printf("Element found in position =

%d\n",succ+1); else

printf("Element not found \n");

}

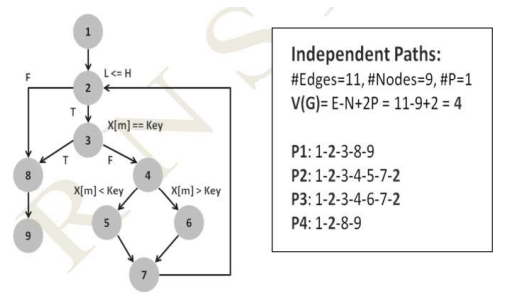
else

printf("Number of element should be greater than zero\n");

return 0;

}

**Grafik Program**

****

Gambar 1. 4 Grafik Program

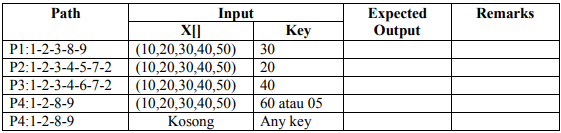
**Test Case**

**Pre-Kondisi:**

*Array* mempunyai elemen2 yang urutannya naik **T/F**

Elemen *Key* ada di dalam *Array* **T/F**

*Array* mempunyai jumlah elemen ganjil **T/F**

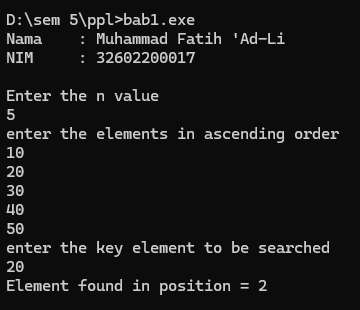


Gambar 1. 5 Tabel Pre-Kondisi

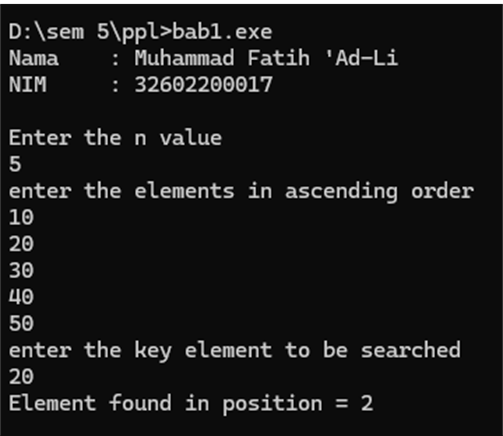
## Hasil Praktikum

Table 1. 1 Tabel Hasil Praktikum

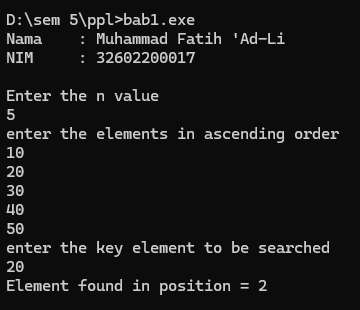
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | ***Path*** | ***Input*** | | ***Expected***  ***Output*** | ***Remarks*** |
| **X[]** | ***Key*** |
| 1 | P1:1-2-3-8-  9 | (10,20,3  0,40,50) | 30 | 3 | 3 (Pass) |
| 2 | P2:1-2-3-4-  5-7-2 | (10,20,3  0,40,50) | 20 | 2 | 2 (pass) |
| 3 | P3:1-2-3-4-  6-7-2 | (10,20,3  0,40,50) | 40 | 4 | 4 (pass) |
| 4 | P4:1-2-8-9 | (10,20,3  0,40,50) | 60 | -1 | -1 (pass) |
| 5 | P4:1-2-8-9 | kosong | Any key | Number of element should be greater than  zero | Number of element should be greater than  zero (pass) |



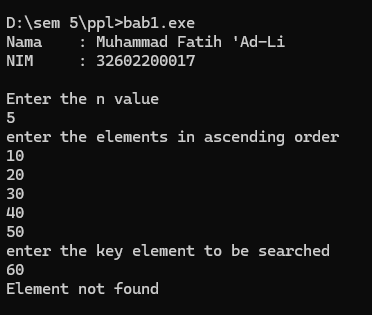
Gambar 1. 6 Hasil Uji 1



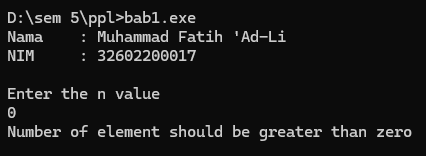
Gambar 1. 7 Hasil Uji 2



Gambar 1. 8 Hasil Uji 3



Gambar 1. 9 Hasil Uji 4



Gambar 1. 10 Hasil Uji 5

## Analisa

Dalam percobaan praktikum bab 1 ini, menggunakan pengujian *whitebox* dengan teknik *basis path* untuk mengukur kompleksitas logis dari desain prosedural dan menggunakannya sebagai pedoman untuk menerapkan himpunan basis dari semua jalur eksekusi. penerapan *basis* *path* terlebih dahulu dari *flowchart/ flowgraph*.

1. Analisa Pengujian Nomor 1

Pada pengujian nomer 1, diperlihatkan mengenai pengujian melalui jalur independen yaitu 1-2-3-8-9. Sebagai contoh dimasukan data *array* X = [10,20,30,40,50] dan kunci yang akan dicari memiliki kunci 30. Proses mencari *key* dimulai dari posisi *array* tengah, maka *function* binsrc langsung melakukan pengembalian nilai posisi *array* di angka 2 ke *function* main. Pada *function* main ini nilai balik ditambah satu, sehingga akan memberikan *output* ke layar yaitu nilai 3.

1. Analisa Pengujian Nomor 2

Pada pengujian nomer 2 diperlihatkan mengenai pengujian melalui jalur independen yaitu 1-2-3-4-6-7-2. Sebagi conntoh dimasukan data *array* X = [10,20,30,40,50] dan *key* yang akan dicari memiliki nilai 20. Pada *function* binsrc mula - mula mendeklarasikan variabel *mid* (*node* 1), setelah itu masuk pada proses perulangan (*node* 2) karena memenuhi syarat perulangan yakni *low* <= *high* dengan nilai *low* adalah 0 dan *high* adalah 4, kemudian mendapatkan nilai *mid* dari operasi *low* + *high* / 2 (*node* 3), kemudian akan memeriksa apakah nilai tengah *array* X sama dengan nilai *key* (*node* 3), karena nilai tidak benar maka lanjut ke *node* 4 untuk memeriksa apakah nilai X urutan *mid* lebih kecil daripada nilai *key*, karena pada kondisi tersebut bernilai salah maka proses berlanjut ke *statement* else yaitu variabel *low* diisi nilai variabel *mid* - 1 (*node* 6). Setelah itu program akan membaca pada *node* 7 dan mengulang kembali ke *node* 2. Ketika kembali ke *node* 2, maka terjadi proses perulangan hingga menemukan posisi *array* yang memiliki nilai sama dengan nilai *key*, yaitu pada posisi *array* *indeks* ke-1. Ketika nilai posisi *array* dikembalikan pada program utama akan ditambahkan 1, sehingga pada layar akan muncul pada posisi 2.

1. Analisa Pengujian Nomor 3

Pada pengujian nomer 3 diperlihatkan mengenai pengujian melalui jalur independen yaitu 1-2-3-4-5-7-2. Sebagi contoh dimasukan data *array* X = [10,20,30,40,50] dan *key* yang akan dicari memiliki nilai 40. Pada *function binsrc* mula - mula mendeklarasikan variabel *mid* (*node* 1), setelah itu masuk pada proses perulangan (*node* 2) karena memenuhi syarat perulangan yaitu *low* <= *high* dengan nilai *low* adalah 0 dan *high* adalah 4, kemudian mendapatkan nilai *mid* dari operasi *low* + *high* / 2 (*node* 3), kemudian akan memeriksa apakah nilai tenah *array* X sama dengan nilai *key* (*node* 3), karena nilai tidak benar maka lanjut ke *node* 4 untuk memeriksa apakah nilai X urutan *mid* lebih kecil daripada nilai *key*, karena pada kondisi tersebut bernilai benar, maka variabel *low* diisi nilai variabel *mid* + 1 (*node* 5). Setelah itu program akan membaca pada *node* 7 dan mengulang kembali ke *node* 2. Ketika kembali ke *node* 2, maka terjadi proses perulangan hingga menemukan posisi *array* yang memiliki nilai sama dengan nilai *key*, yaitu pada posisi *array* *indeks* ke-3. Ketika nilai posisi *array* dikembalikan pada program utama akan ditambahkan 1, sehingga pada layar akan muncul pada posisi 4

1. Analisa Pengujian Nomor 4

Pada pengujian nomer 4 diperlihatkan mengenai pengujian melalui jalur *another path* yaitu 1-2-8-9. Sebagai contoh dimasukan data *array* X = [10,20,30,40,50] dan *key* yang akan dicari memiliki nilai 60. Pada *function* binsrc mula - mula mendeklarasikan variabel *mid* (*node* 1), setelah itu masuk pada proses perulangan (*node* 2) karena memenuhi syarat perulangan yaitu *low <= high* dengan nilai *low* adalah 0 dan *high* adalah 4, kemudian mendapatkan nilai *mid* dari operasi *low* + *high* / 2 (*node* 3), kemudian akan memeriksa apakah nilai tenah *array* X sama dengan nilai *key* (*node* 3), karena nilai tidak benar maka lanjut ke *node* 4 untuk memeriksa apakah nilai X urutan *mid* lebih kecil daripada nilai *key*, karena pada kondisi tersebut bernilai benar, maka variabel *low* diisi nilai variabel *mid* + 1 (*node* 5). Setelah itu program akan membaca pada *node* 7 dan mengulang kembali ke *node* 2. Ketika kembali ke *node* 2, maka terjadi proses perulangan hingga suatu kondisi dimana perulangan berhenti, karena *low* > *high*, dan program akan mengembalikan nilai -1 (*node* 8) dan proses eksekusi pada *function* binsrc berakhir (*node* 9). Ketika nilainya dikembalikan pada program utama akan di cek apakah nilainya lebih dari sama dengan 0, karena lebih kecil dari 0, maka pada layar akan muncul pada posisi -1.

1. Analisa Pengujian Nomor 5

Pada pengujian nomer 5, jumlah *array* yang dimasukan adalah 0. Kemudian dilakukan pengecekan apakah jumlah *array* yang dimasukan lebih dari 0. Karena jumlah *array* yang dimasukan sama dengan 0, maka program akan mengeksekusi *statement else* yaitu menampilkan *output* pada *console* “*Number of element should be greater than zero*”, yang artinya elemen pada *array* tidak boleh sama dengan 0 (kosong). Sehingga pada pengujian Nomor 5 *function* binsrc tidak dieksekusi. Dari keseluruhan analisa di atas, kita berhasil melakukan kelima pengujian dengan sukses. Karena semua ekspektasi kita harapkan sesuai dengan hasil yang ditampilkan

## Kesimpulan

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merupakan *review* akhir dari spesifikasi, perancangan dan pengkodean. pengujian dilakukan dengan mendemonstasikan fungsi tersebut bisa berjalan dengan sempurna atau ada *error* (*Black box Testing*) maupun dengan mengetahui cara kerja internal dari produk tersebut (*White Box Testing).*

Pada bab 1 ini, kita belajar mengenai pengujian perangkat lunak menggunakan metode *White Box* berdasarkan pengujian *Basis path*. Metode *Basis path Testing* membuat perancang kasus uji bisa menurunkan sebuah ukuran kompleksitas logika dari sebuah perancangan prosedural dan ukuran ini selanjutnya digunakan untuk menentukan sekelompok data dasar (basis set) dari jalur eksekusi.

**BAB II  
PENGUJIAN *WHITEBOX* (2): PENGUJIAN KONDISI**

****

****

# BAB II PENGUJIAN *WHITEBOX* (2): PENGUJIAN KONDISI

1. **Tujuan**

Adapun tujuan praktikum ini, adalah :

1. Praktikan dapat mempersiapkan tahapan pengujian.
2. Praktikan dapat merancang kasus uji.
3. Praktikan dapat melakukan pengujian-pengujian perangkat lunak dengan teknik pengujian kondisi (menggunakan tabel keputusan).
4. **Alat dan Bahan**

Alat dan bahan yang dibutuhkan dalam modul bab 2 ini adalah sebuah komputer yang dilengkapi dengan *“CodeBlock” IDE* (*Integrated Development Environtmen*) yang memiliki *Compiler* C.

1. **Dasar Teori**
2. **Teknik Pengujian Perangkat Lunak**

Pengujian kondisi merupakan metode desain *test case* yang menguji  
kondisi logika dalam suatu perangkat lunak. Pengujian ini dimaksudkan  
untuk mendeteksi *error* kondisi pada suatu kondisi perangkat lunak dan  
mendeteksi *error* lainnya dan mempunyai keuntungan sebagai berikut :

1. Pengukuran jangkauan pengujian kondisi adalah sederhana.
2. Jangkauan pengujian kondisi pada perangkat lunak menyediakan  
   petunjuk pembangkitan pengujian tambahan bagi perangkat lunak.  
   Contoh: kondisi sederhana dari persamaan relasional

**E1 (Operator relasional) E2**

E1 dan E2 merupakan persamaan matematika

Operator Relasional adalah salah satu dari operator berikut ini :

<, ≤, =, ≠ ( - = ), >, ≥

Operator *Boolean* : OR (‘│’), AND (‘&’), NOT (‘-‘)

Yang termasuk strategi pengujian kondisi adalah :

1. Pengujian cabang

Merupakan strategi penujian kondisi yang paling sederhana. Untuk  
mencapai suatu kondisi gabungan C, cabang-cabang *True* dan *False*dari C dan setiap kondisi pada C perlu dieksekusi paling tidak satu  
kali.

1. Pengujian domain

Membutuhkan tiga atau empat pengujian yang dilakukan untuk sebuah persamaan relasional.

**E1 > E2, E1 = E2, E1 < E2**

1. Pengujian BRO (*Branch and Relational Operator* )

Menggunakan batasan kondisi C. batasan kondisi C dengan n kondisi  
sederhana (D1, D2, …, Dn) dimana Di(0 < i ≤ n) merupakan batasan  
akhir dari kondisi C.

Contoh 1 :

**C2 : B1 & (E3 = E4)**

B1 adalah variabel *Boolean*.

E3 dan E4 adalah persamaan matematika atau aritmatika.  
Batasan kondisi C2 adalah bentuk (D1, D2) dengan D1 adalah ‘t’ atau  
‘f’, dan D2 adalah ‘>’, ‘=’, ‘<’

C2 adalah persamaan reasional dengan memodifikasi himpunan  
pembatas {(t, t), (f, t), (t, f)}.

‘t’ untuk (E3=E4) mengimplikasikan ‘=’

‘f ‘ untuk (E3=E4) mengimplikasikan ‘<’ atau ‘>’

Dengan menggantikan

**(t,t) 🡪 (t, =)**

**(f,t) 🡪 (f, =)**

**(t,f) 🡪 (t, <) atau (t, <>) atau (t, >)**

Maka hasil himpunan Batasan untuk C2 adalah :

**{ (t, =), (f, =), (t, >), (t, <) }**

1. **Prosedur Praktikum**

Berikut ini merupakan kode program dalam bahasa C untuk masalah segitiga yaitu menerima tiga *integer* yang dianggap sebagai tiga sisi segitiga dan menentukan apakah ketiga nilai tersebut merepresentasikan segitiga sama sisi, segitiga sama kaki, segitiga yang tidak sama semua sisinya atau tidak membentuk segitiga.

1. Menentukan *Basis Path*.
2. Menurunkan *Test Case*-nya.
3. Menjalankan *Test Case* tersebut.
4. Menganalisis hasil pengujian tersebut.

**Program segitiga**

#include <stdio.h>

#include <stdlib.h>

int main()

{

printf("Nama\t: Muhammad Fatih ‘Ad-Li\n");

printf("NIM\t: 32602200017\n\n");

int a,b,c,c1,c2,c3;

char is*triangle*;

do

{

printf("\nenter 3 integers which are sides of

*triangle*\n");

scanf("%d%d%d",&a,&b,&c);

printf("\na=%d\tb=%d\tc=%d",a,b,c);

c1 = a>=1 && a<=10;

c2= b>=1 && b<=10;

c3= c>=1 && c<=10;

if (!c1)

printf("\nthe value of a=%d is not the

range of permitted value",a);

if (!c2)

printf("\nthe value of b=%d is not the

range of permitted value",b);

if (!c3)

printf("\nthe value of c=%d is not the

range of permitted value”,c);

}

while(!(c1 && c2 && c3));

// Mengecek segitiga atau tidak

if( a<b+c && b<a+c && c<a+b )

is*triangle*='y';

else

is*triangle* ='n';

if (is*triangle*=='y')

if ((a==b) && (b==c))

printf("\nEquilateral *triangle*\n");

else if ((a!=b) && (a!=c) && (b!=c))

printf("\nScalene *triangle*\n");

else

printf("\nIsosceles *triangle*\n");

else

printf("\nNot a *triangle*\n");

return 0;

}

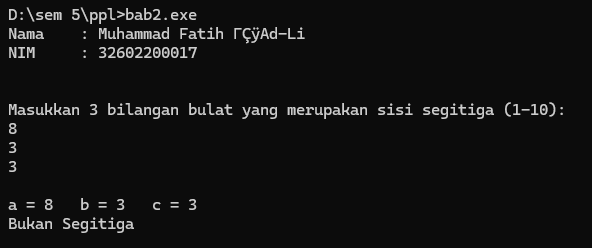
***Test Case***

1. *Test* data : memasukkan 3 nilai *integer* (a, b, dan c)
2. *Pre-condition* : a < b + c, b < a + c dan c < a + b
3. Deskripsi singkat : cek apakah nilai merupakan segitiga sama sisi,  
   segitiga sama kaki, segitiga yang tidak semua sama sisinya atau tidak  
   membentuk segitiga.

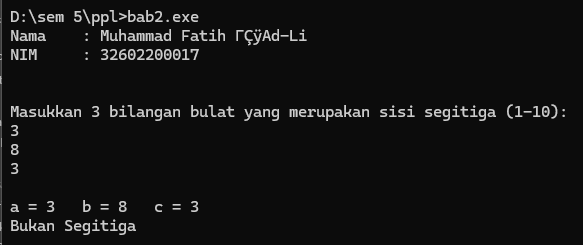
Table 2. 1 Tabel Keputusan atau Kondisi

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *RULES* | | R  1 | R  2 | R  3 | R  4 | R  5 | R  6 | R  7 | R  8 | R  9 | R  10 | R  11 |
| *Conditions* | C1: a < b + c | F | T | T | T | T | T | T | T | T | T | T |
| C2: b < a + c | - | F | T | T | T | T | T | T | T | T | T |
| C3: c < a + b | - | - | F | T | T | T | T | T | T | T | T |
| C4: a = b | - | - | - | T | T | T | T | F | F | F | F |
| C5: a = c | - | - | - | T | T | F | F | T | T | F | F |
| C6: b = c | - | - | - | T | F | T | F | T | F | T | F |
| *Actions* | A1: *Not a triangle* | X | X | X |  |  |  |  |  |  |  |  |
| A2: *Scalene triangle* |  |  |  |  |  |  |  |  |  |  | X |
| A3: *Isosceles triangle* |  |  |  |  |  |  | X |  | X | X |  |
| A4: *Equational triangle* |  |  |  | X |  |  |  |  |  |  |  |
| A5: *Impossible* |  |  |  |  | X | X |  | X |  |  |  |

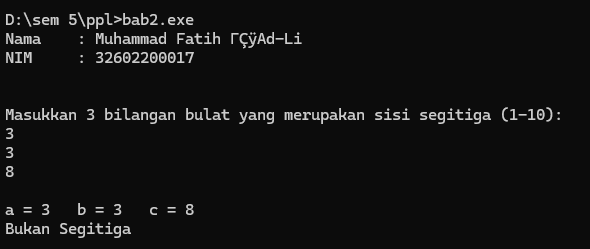
1. **Hasil Praktikum**
2. **Pengujian Sesuai Tabel Keputusan**
3. *Output*



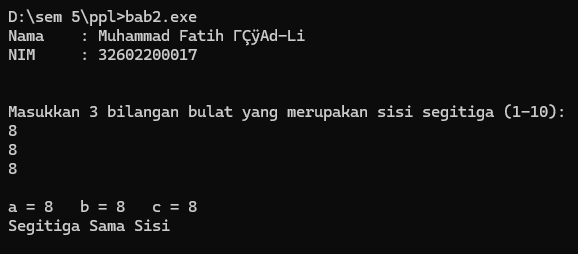
Gambar 2. 1 Output Hasil Pengujian Rules 1



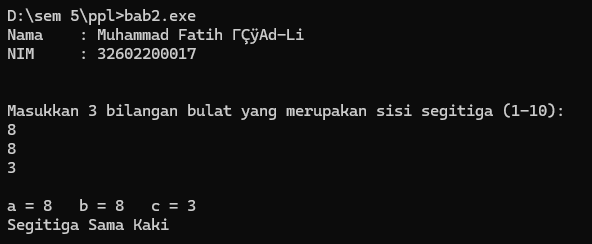
Gambar 2. 2 Output Hasil Pengujian Rules 2



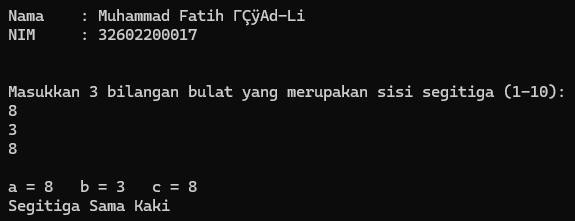
Gambar 2. 3 Output Hasil Pengujian Rules 3



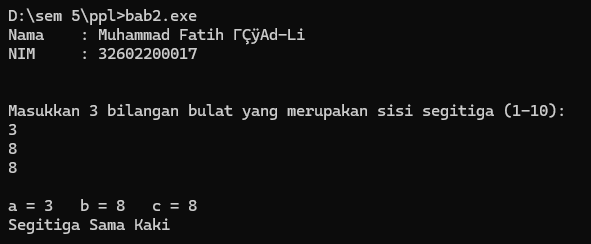
Gambar 2. 4 Output Hasil Pengujian Rules 4



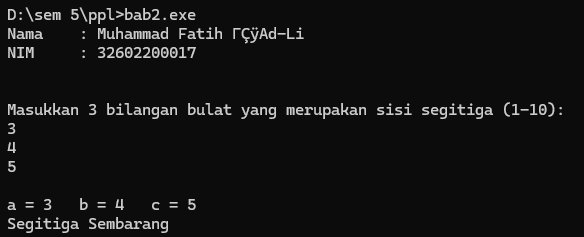
Gambar 2. 5 Output Hasil Pengujian Rules 7



Gambar 2. 6 Output Hasil Pengujian Rules 9



Gambar 2. 7 Output Hasil Pengujian Rules 10



Gambar 2. 8 Output Hasil Pengujian Rules 11

1. Penjelasan

Pengujian perangkat lunak yang menguji dengan menggunakan kondisi kombinasi dari pengujian cabang dan pengujian domain untuk menerima tiga *integer* sebagai tiga sisi segitiga sesuai dengan *rules* (aturan) dari nomor 1 sampai 11. Namun, pada *rules* nomor 5, 6, dan 8 tidak memiliki *output*, dikarenakan tidak diketahui berapa sisi-sisi yang di*input*kan kedalam *source code*. Untuk itu, dalam hasil praktikum ini *output* yang ditampilkan seperti yang ditunjukkan pada gambar 2.1 sampai dengan gambar 2.8 dengan *source code* bahasa C yang dijalankan dengan aplikasi *CodeBlocks*. Sehingga hasil dari percobaan pengujian perangkat lunak yang telah dijalankan dapat kita tuliskan pada tabel hasil uji berikut.

Table 2. 2 Hasil Pengujian Kondisi

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rules | | | C1 | C2 | C3 | C4 | C5 | C6 | Output yg diharapkan |
| a<b+c | b<a+c | c<a+b | a=b | a=c | b=c |
| R1 | a | 8 | F | - | - | - | - | - | Not a triangle |
| b | 3 |
| c | 3 |
| R2 | a | 3 | T | F | - | - | - | - | Not a triangle |
| b | 8 |
| c | 3 |
| R3 | a | 3 | T | T | F | - | - | - | Not triangle |
| b | 3 |
| c | 8 |
| R4 | a | 8 | T | T | T | T | T | T | Equilateral triangle |
| b | 8 |
| c | 8 |
| R5 | A |  | T | T | T | T | T | F | Imposibble |
| B |  |
| C |  |
| R6 | A |  | T | T | T | T | F | T | Imposibble |
| B |  |
| C |  |
| R7 | A | 8 | T | T | T | T | F | F | Isosceles triangle |
| B | 8 |
| C | 3 |
| R8 | A |  | T | T | T | F | T | T | Imposibble |
| B |  |
| C |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R9 | a | 8 | T | T | T | F | T | F | Isosceles triangle |
| b | 3 |
| c | 8 |
| R10 | a | 3 | T | T | T | F | F | T | Isosceles triangle |
| b | 8 |
| c | 8 |
| R11 | a | 3 | T | T | T | F | F | F | Scalene triangle |
| b | 4 |
| c | 5 |

1. **Analisa**

Dari hasil percobaan praktikum dengan teknik pengujian kondisi untuk mendeteksi *error* kondisi dari hasil percobaan praktikum yang telah kita lakukan yakni, mendeteksi dan mengklasifikasikan jenis segitiga berdasarkan tabel keputusan pada modul, maka didapatkan analisanya sebagai berikut :

1. Analisa Pengujian Kondisi *Rules* 1

Pada pengujian *rules* 1, pada gambar 2.1 diperlihatkan diberikan nilai a = 8, b = 3, c = 3. Selanjutnya dijalankan sesuai kondisi C1 yaitu a < b + c yang berfungsi untuk mengetahui apakah nilai a,b,c termasuk segitiga atau tidak. Maka 8 < 3 +3 dinyatakan salah karena 8 < 6 (8 kurang dari 6) dan kemudian program mengeluarkan *output* berupa *not triangle* yang berarti *rules* 1 menunjukkan bahwa kondisi tersebut bukan termasuk segitiga.

1. Analisa Pengujian Kondisi *Rules* 2

Sebagai contoh pengujian *rules* 2, , pada gambar 2.2 diperlihatkan diberikan nilai a = 3, b = 8, c = 3. Selanjutnya dijalankan sesuai kondisi C1: yaitu a < b + c dan C2: b < a + c. Pada pengujian ini 3 < 8 + 3 atau 3 < 12 hal ini memenuhi kondisi C1 dan benar. Kemudian melanjutkan ke kondisi C2 yaitu b < a + c. Pada pengujian ini 8 < 3 + 3 atau 8 < 6 hal ini tidak memenuhi kondisi C2 atau salah. Maka *output* pada program akan muncul *not a triangle* yang berarti *rules* 2 menunjukkan bahwa kondisi tersebut bukan termasuk segitiga.

1. Analisa Pengujian Kondisi *Rules* 3

Pada hasil pengujian kondisi *rules* 3, pada gambar 2.3 diperlihatkan diberikan nilai a = 3, b = 3, c = 8. Pada pengujian *rule* 3 harus memenuhi tiga kondisi yaitu C1: a < b + c, C2: b < a + c, dan C3: c < a + b. Nilai a, b, c diperiksa pada perintah *if*, apakah memenuhi ketentuan kondisi C1 dan C2 tetapi tidak memenuhi kondisi dari C3. Pada C1: 3 < 3 + 8 dimana 3 < 11 menghasilkan nilai *true* atau terpenuhi, kemudian untuk C2: 3 < 3 + 8 dimana 3 < 11 menghasilkan nilai *true* dan kemudian untuk C3: 8 < 3 + 3 dimana 8 < 6 dan menghasilkan nilai *false* atau tidak terpenuhi maka program akan menghasilkan *action* yang berupa *not a triangle* yang berarti *rules* 3 menunjukkan bahwa kondisi tersebut bukan termasuk segitiga.

1. Analisa Pengujian Kondisi *Rules* 4

Pada pengujian selanjutnya yaitu kondisi *rules* 4, , pada gambar 2.4 ditampilkannya nilai variabel a = 8, b = 8, c = 8. Pada C1: 8 < 8 + 8 menghasilkan nilai *true* (memenuhi), sedangkan C2: 8 < 8 + 8 menghasilkan nilai *true* (memenuhi), C3: 8 < 8 + 8 menghasilkan nilai *true* (memenuhi), C4: 8=8 menghasilkan nilai *true* (memenuhi), C5: 8=8 menghasilkan nilai *true* (memenuhi), dan C6: 8=8 menghasilkan nilai *true* (memenuhi). maka kondisi percabangan tersebut terpenuhi semua dan menampilkan *action* bahwa jenis segitiga tersebut adalah *equilateral triangle* yang berarti *rules* 4 menunjukkan bahwa kondisi tersebut membentuk segitiga sama sisi.

1. Analisa Pengujian Kondisi *Rules* 5

Pada pengujian *rules* 5, , pada gambar 2.5 memiliki ketentuan untuk memenuhi enam kondisi, yakni C1: a < b + c memenuhi (*true*), C2: b < a + c memenuhi (*True)*, C3: c < a + b memenuhi (*true*), C4: a = b memenuhi (*true*), C5: a = c memenuhi (*true*), dan C6: b = c tidak memenuhi (*false*). Apabila tidak ada nilai yang memenuhi untuk kondisi C4, C5, C6, maka jenis segitiga tersebut adalah *impossible*, yang berarti *rules* 5 menunjukkan bahwa kondisi tersebut tidak memungkinkan untuk terbentuk sebuah segitiga.

1. Analisa Pengujian Kondisi *Rules* 6

Pada pengujian *rule* 6, pada gambar 2.6 diperlihatkan harus memenuhi enam kondisi yaitu C1 a<b+c memenuhi (*true*), C2 b<a+c memenuhi (*true*), C3 c<a+b memenuhi (*true*), C4 a=b memenuhi (*true*), C5 a=c tidak terpenuhi (*false*), dan C6 b=c memenuhi (*true*). Karena tidak ada nilai yang memenuhi untuk kondisi C4, C5, C6, dimana nilai a tidak boleh sama dengan nilai c, namun nilai a harus sama dengan b dan nilai b sama dengan nilai c, maka jenis segitiga tersebut adalah *impossible.*

1. Analisa Pengujian Kondisi *Rules* 7

Pada pengujian *rule* 7, pada gambar 2.7 diperlihatkan harus memenuhi enam kondisi yaitu C1 a<b+c memenuhi (*true*), C2 b<a+c memenuhi (*true*), C3 c<a+b memenuhi (*true*), C4 a=b memenuhi (*true*), dan untuk C5 a=c tidak terpenuhi (*false*), dan C6 b=c tidak terpenuhi (*false*). maka diberikan nilai a=8, b=8, c=3. Dari kondisi C1 8<8+3 maka bernilai *true*, kondisi C2 8<8+3 maka bernilai *true*, kondisi C3 8<8+8 maka bernilai *true.* Nilai C1, C2, dan C3 terpenuhi semua (*true*) jika dioperasikan dengan operator logika *AND* maka akan menghasilkan nilai benar, sehingga variabel *istriangle* akan diisi karakter ‘y’. Sedangkan kondisi C4 8=8 maka bernilai *true*, kondisi C5 8=3 maka bernilai *false* karena 8≠3, begitu juga dengan C6 8=3 maka bernilai *false* karena 8≠3. sehingga program akan menampilkan *output* berupa *Isosceles Triangle* (segitiga sama kaki).

1. Analisa Pengujian Kondisi *Rules* 8

Pada pengujian *rule* 8, pada gambar 2.8 diperlihatkan harus memenuhi enam kondisi C1 a<b+c memenuhi (*true*), C2 b<a+c memenuhi (*true*), C3 c<a+b memenuhi (*true*), C4 a=b tidak terpenuhi (*false*), C5 a=c memenuhi (*true*), dan C6 b=c memenuhi (*true*). Karena tidak ada nilai yang memenuhi untuk kondisi C4, C5, C6, dimana nilai a tidak boleh sama dengan nilai b, namun nilai b harus sama dengan c dan nilai a sama dengan nilai c, maka jenis segitiga tersebut adalah *impossible.*

1. Analisa Pengujian Kondisi *Rules* 9

Pada pengujian *rule* 9, pada gambar 2.9 diperlihatkan harus memenuhi enam kondisi yaitu C1 a<b+c memenuhi (*true*), C2 b <a+c memenuhi (*true*), C3 c<a+b memenuhi (*true*), C4 a=b tidak terpenuhi (*false*), C5 a=c memenuhi (*true*), C6 b=c tidak terpenuhi (*false*). maka nilai yang dapat memenuhi adalah a=8, b=3, c=8. Pada percabangan pertama kondisi C1 8<3+8 adalah *true*, kondisi C2 3<8+8 adalah *true*, dan kondisi C3 8<8+3 adalah *true*. Nilai C1, C2, dan C3 terpenuhi semua (*true*) jika dioperasikan dengan operator logika *AND* maka akan menghasilkan nilai benar, sehingga variabel *istriangle* akan diisi karakter ‘y’. sedangkan untuk C4 8=3 adalah *false* karena 8≠3, C5 8=8 adalah *true*, dan untuk C6 8=3 adalah bernilai *false* karena 8≠3. yang berarti *rules* 9 menunjukkan bahwa kondisi tersebut membentuk segitiga sama kaki. sehingga program akan menampilkan *output* berupa *Isosceles Triangle* (segitiga sama kaki)*.*

1. Analisa Pengujian Kondisi *Rules* 10

Pada pengujian *rule* 10, pada gambar 2.10 diperlihatkan harus memenuhi enam kondisi yaitu C1 a<b+c memenuhi (*true*), C2 b<a+c memenuhi (*true*), C3 c<a+b memenuhi (*true*), C4 a=b tidak terpenuhi (*false*), C5 a=c tidak memenuhi (*false*), C6 b=c memenuhi (*true*). maka nilai yang dapat memenuhi adalah a=3, b=8, c=8. Pada percabangan pertama kondisi C1 3<8+8 adalah *true*, kondisi C2 <3+8 adalah *true*, dan kondisi C3 8<3+8 adalah *true*. Nilai C1, C2, dan C3 terpenuhi semua *true,* jika dioperasikan dengan operator logika *AND* maka akan menghasilkan nilai benar, sehingga variabel *istriangle* akan diisi karakter ‘y’, sedangkan untuk C4 3=8 adalah *false* karena 3≠8, C5 3=8 adalah *false* karena 3≠8, dan untuk C6 8=8 terpenuhi atau *true*. sehingga program akan menampilkan *output* berupa *Isosceles Triangle* (segitiga sama kaki)*.*

1. Analisa Pengujian Kondisi *Rules* 11

Pada pengujian *rule* 11, pada gambar 2.11 diperlihatkan harus memenuhi enam kondisi yaitu C1 a<b+c memenuhi (*true*), C2 b<a+c memenuhi (*true*), C3 c<a+b memenuhi (*true*), C4 a=b tidak terpenuhi (*false*), C5 a=c tidak terpenuhi (*false*), dan C6 b=c tidak terpenuhi (*false*). Diberikan nilai variabel a=3, b=4, dan c=5. Pada percabangan pertama diberikan kondisi C1 3<4+5 adalah *true*, C2 4<3+5 adalah *true*, dan C3 5<3+4 adalah *true*, Sehingga nilai C1, C2, dan C3 *true*, *true*, *true* yang dioperasikan dengan operator *AND* akan menghasilkan nilai benar dan variabel *istriangle* akan diisi karakter ‘y’. sedangkan untuk C4 3=4 adalah *false* karena 3≠4, C5 3=5 adalah *false* karena 3≠5, dan untuk C6 4=5 tidak terpenuhi atau *false* karena 4≠5. sehingga program akan meampilkan *output* berupa *Scalene Triangle* (segitiga sembarang)*.*

1. **Kesimpulan**

Berdasarkan hasil praktikum Bab II diatas, menjelaskan mengenai pengujian *white box* suatu perangkat lunak untuk menjamin operasi-operasi *internal* sesuai dengan spesifikasi yang telah di tetapkan dengan menggunakan struktur yang telah di rancang, dengan menggunakan kondisi kombinasi dari pengujian cabang dan pengujian domain.

Metode ini membantu mengidentifikasi potensi kesalahan dan memastikan kehandalan sistem dalam mengatasi variasi kondisi yang mungkin terjadi. Dengan melakukan pengujiann kondisi secara menyeluruh, tim pengembangan dapat memastikan bahwa perangkat lunak.

Disamping itu, menguji perhitungan dengan menggunakan operator relasional dan operator *boolean,* untukmenerima tiga *integer* yang dianggap sebagai tiga sisi segitiga untuk mendapatkan *action* jenis-jenis segitiga sebagai *output* yang diantarnya, adalah segitiga sama sisi, segitiga sama kaki, segitiga yang tidak sama semua sisinya, atau tidak membentuk segitiga.

**BAB III  
PENGUJIAN *BLACKBOX* (1):   
PENGUJIAN ANALISIS KELAS EKUIVALENSI**

****

****

# 

# BAB III PENGUJIAN *BLACKBOX* (1): PENGUJIAN ANALISIS KELAS EKUIVALENSI

1. **Tujuan**

Adapun tujuan praktikum ini, adalah :

1. Praktikan dapat mempersiapkan tahapan pengujian.
2. Praktikan dapat merancang kasus uji.
3. Praktikan dapat melakukan pengujian perangkat lunak dengan teknik Analisis Kelas Ekuivalensi.
4. **Alat dan Bahan**

Alat dan bahan yang dibutuhkan dalam modul bab 3 ini adalah sebuah komputer yang dilengkapi dengan “*CodeBlock*” IDE (*Integrated Development Environtmen*) yang memiliki *Compiler* C.

1. **Dasar Teori**
2. **Pengujian *Black Box***

Pengujian *black box* (*black box testing*) memfokuskan pada kebutuhan fungsional dari *software*. Hal ini berarti bahwa pengujian ini memperbolehkan *softwareengineer* menurunkan sejumlah *input* yang ditujukan untuk menguji kebutuhan fungsional dari program tersebut. Pengujian ini berusaha menemukan *error* dengan kategori sebagai berikut :

1. Fungsi yang salah atau hilang.
2. Kesalahan antarmuka, struktur data atau pengaksesan data eksternal, untuk kerja, inisialisasi dan penghentian.

Tidak seperti pengujian *white box* yang dilakukan pada awal proses,  
pengujian *black box* diterapkan pada akhir tahapan proses pengujian. Hal ini  
dikarenakan pengujian ini tidak mementingkan struktur kontrol tetapi lebih  
memfokuskan pada domain informasi.

1. **Analisis Partisi Kelas Ekuivslensi (*Equivalence Class Partitioning*)**

Merupakan metode pengujian *black box* yang membagi *input* domain dari program kedalam kelas-kelas data dimana kasus uji bisa diturunkan. *Equivalence partitioning* didasarkan pada evaluasi persamaan kelas dari  
*input condition*. Sebuah persamaan kelas menunjukkan sekumpulan keadaan  
valid atau tidak valid untuk syarat atau kondisi masukan yang umumnya  
adalah nilai numerik tertentu, sebuah jangkauan nilai (*range value*), sebuah  
himpunan nilai-nilai yang berkaitan, atau kondisi boolean. Persamaan kelas  
bisa didefinisikan menurut panduan sebagai berikut :

1. Jika kondisi *input* adalah sebuah *range*, satu kelas persamaan dan  
   dua kelas persamaan didefinisikan.
2. Jika sebuah kondisi *input* memerlukan sebuah nilai khusus, satu  
   kelas persamaan dan dua kelas persamaan didefinisikan.
3. Jika sebuah kondisi *input* adalah sebuah anggota himpunan, satu  
   kelas persamaan, satu kelas persamaan yang valid dan satu kelas persamaan yang tidak valid didefinisikan.
4. Jika kondisi *input* adalah boolean, satu kelas yang validdan satu  
   kelas yang tidak valid.

Contoh :

Pemeliharaan data untuk aplikasi bank yang sudah diotomatiskan.  
Pemakai dapat memutar nomor telepon bank dengan menggunakan  
mikro komputer yang terhubung dengan *password* yang telah  
ditentukan dan diikuti dengan perintah-perintah. Data yang diterima  
adalah sebagai berikut :

Kode Area : Kosong atau tiga digit

*Prefix* : 3 digit atau tidak diawali 0 atau 1

*Suffix* : 4 digit

*Password* : 6 digit alfanumerik

Perintah : *check*, *deposit*,dan lain-lain.

Selanjutnya kondisi *input* digabungkan dengan masing-masing data  
elemen dapat ditentukan sebagai berikut :

Kode Area : Kondisi *input*, *boolean* – kode area mungkin ada atau tidak

Kondisi *input*, *range* – nilai ditentukan antara 200 dan 999

*Prefix* : Kondisi *inputrange* > 200 atau tidak diawali 0 atau 1

*Suffix* : Kondisi *input* nilai 4 digit

*Password* : Kondisi *input* *boolean* – *pw* mungkin diperlukan atau tidak

Kondisi *input* nilai dengan 6 karakter *string*

Perintah : Kondisi *input* set berisi perintah-perintah yang telah

didefinisikan

1. **Prosedur Praktikum**

Berikut ini merupakan kode program dalam bahasa C untuk masalah segitiga yaitu menerima tiga *integer* yang dianggap sebagai tiga sisi segitiga dan menentukan apakah ketiga nilai tersebut merepresentasikan segitiga sama sisi, segitiga sama kaki, segitiga yang tidak sama semua sisinya atau tidak membentuk segitiga.

1. Menentukan *Basis Path*.
2. Menurunkan *Test Case*-nya.
3. Menjalankan *Test Case* tersebut.
4. Menganalisis hasil pengujian tersebut.

Program segitiga

#include <stdio.h>

#include <stdlib.h>

int main()

{

printf("Nama\t: Muhammad Fatih ‘Ad-Li\n");

printf("NIM\t: 32602200017\n\n");

int a,b,c,c1,c2,c3;

char istriangle;

do

{

printf("\nenter 3 integers which are sides of

triangle\n");

scanf("%d%d%d",&a,&b,&c);

printf("\na=%d\tb=%d\tc=%d",a,b,c);

c1 = a>=1 && a<=10;

c2= b>=1 && b<=10;

c3= c>=1 && c<=10;

if (!c1)

printf("\nthe value of a=%d is not the

range of permitted value",a);

if (!c2)

printf("\nthe value of b=%d is not the

range of permitted value",b);

if (!c3)

printf("\nthe value of c=%d is not the

range of permitted value”,c);

}

while(!(c1 && c2 && c3));

// Mengecek segitiga atau tidak

if( a<b+c && b<a+c && c<a+b )

istriangle='y';

else

istriangle ='n';

if (istriangle=='y')

if ((a==b) && (b==c))

printf("\nEquilateral triangle\n");

else if ((a!=b) && (a!=c) && (b!=c))

printf("\nScalene triangle\n");

else

printf("\nIsosceles triangle\n");

else

printf("\nNot a triangle\n");

return 0;

}

*Test Case*

1. *Test* data : Memasukkan 3 nilai *integer* (a, b, dan c)
2. *Pre-condition* : 1 ≤ a ≤ 10, 1 ≤ b ≤ 10 dan 1 ≤ c ≤ 10 dan a < b + c, b < a + c dan c < a + b
3. Deskripsi singkat : memeriksa apakah nilai merupakan segitiga sama sisi segitiga sama kaki, segitiga yang tidak sama semua sisinya atau tidak membentuk segitiga.

Table 3. 1 Weak Equivalence Class Testing

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Weak Equivalence Class Testing* | | | | | | | | |
| *Case Id* | *Description* | *Input Data* | | | *Expected Output* | *Actual Output* | *Status* | *Comments* |
| A | B | C |
| 1 | *Enter the normal value for* a, b, *and* c | 5 | 5 | 5 | *Should display the message equilateral triangle* | *Equila-teral triangle* | *Pass* |  |
| 2 | *Enter the normalvalue for* a, b, *and* c | 2 | 2 | 3 | *Should display the message isosceles triangle* | *Isosce-les triangle* | *Pass* |  |
| 3 | *Enter the normalvalue for* a, b, *and* c | 3 | 4 | 5 | *Should display the message scalene triangle* | *Scalene triangle* | *Pass* |  |
| 4 | *Enter the normalvalue for* a, b, *and* c | 4 | 1 | 2 | *Message should be displayed can’t form a triangle* | *Not a triangle* | *Pass* |  |

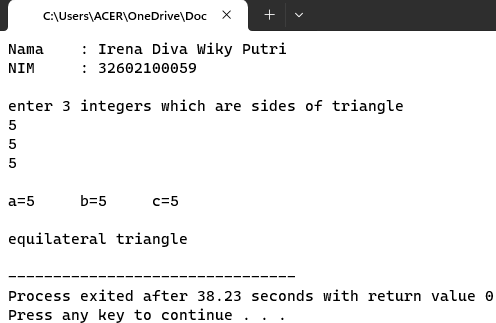
Table 3. 2 Weak Robust Equivalence Class Testing

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Weak Robust Equivalence Class Testing* | | | | | | | | |
| *Case Id* | *Description* | *Input Data* | | | *Expected Output* | *Actual Output* | *Status* | *Comments* |
| A | B | C |
| 5 | *Enter one invalid input (min) value and two valid value for a, b and c* | -1 | 5 | 5 | *Should display value of a is not in the range of permit-ted values* | *Not the range of per-mitted value* | *Pass* |  |
| 6 | *Enter one invalid input (min) value and two valid value for a, b and c* | 5 | -1 | 5 | *Should display value of b is not in the range of permit-ted values* | *Not the range of per-mitted value* | *Pass* |  |
| 7 | *Enter one invalid input (min) value and two valid value for a, b and c* | 5 | 5 | -1 | *Should display value of c is not in the range of permit-ted values* | *Not the range of per-mitted value* | *Pass* |  |
| 8 | *Enter one invalid input (max) value and two valid value for a, b and c* | 11 | 5 | 5 | *Should display value of a is not in the range of permit-ted values* | *Not the range of per-mitted value* | *Pass* |  |
| 9 | *Enter one invalid input (max) value and two valid value for a, b and c* | 5 | 11 | 5 | *Should display value of b is not in the range of permit-ted values* | *Not the range of per-mitted value* | *Pass* |  |
| 10 | *Enter one invalid input (max) value and two valid value for a, b and c* | 5 | 5 | 11 | *Should display value of c is not in the range of permit-ted values* | *Not the range of per-mitted value* | *Pass* |  |

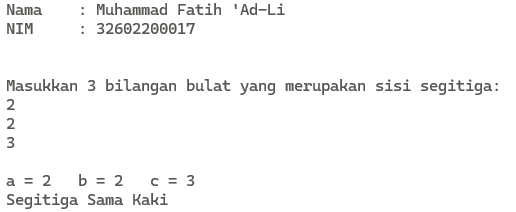
Table 3. 3 Strong Robust Equivalence Class Testing

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Strong Robust Equivalence Class Testing* | | | | | | | | |
| *CaseId* | *Description* | *Input Data* | | | *Expected Output* | *Actual Output* | *Status* | *Comme-nts* |
|  |  |  |
| 11 | *Enter two invalid input (min) and two valid value for a, b and c* | -1 | 3 | 3 | *Should display value of a is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap  kan |
| 12 | *Enter two invalid input (min) and two valid value for a, b and c.* | 3 | -1 | 3 | *Should display value of b is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| 13 | *Enter two invalid input (min) and two valid value for a, b and c.* | 3 | 3 | -1 | *Should display value of c is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| 14 | *Enter all invalid inputs (min)* | -1 | -1 | 5 | *Should display value of a is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| *Should display value of b is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| 15 | *Enter two invalid input (max) and two valid value for a, b and c* | 5 | -1 | -1 | *Should display value of b is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| *Should display value of c is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| 16 | *Enter two invalid input (max) and two valid value for a, b and c* | -1 | 5 | -1 | *Should display value of a is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| *Should display value of c is not the range of*  *permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| 17 | *Enter two invalid input (max) and two valid value for a, b and c* | -1 | -1 | -1 | *Should display value of a is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| *Should display value of b is not the range of permitte*  *d values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |
| *Should display value of c is not the range of permitted values* | *Not the range of permit-ted value* | *pass* | Sesuai yang diharap kan |

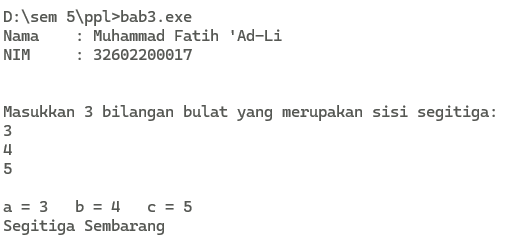
1. **Hasil Praktikum**
2. **Pengujian Sesuai Tabel Keputusan**
3. *Output*



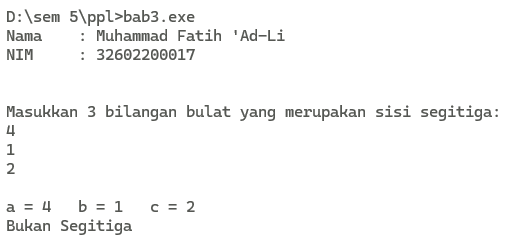
Gambar 3. 1 Output Test Case 1 Black Box Equivalence Class



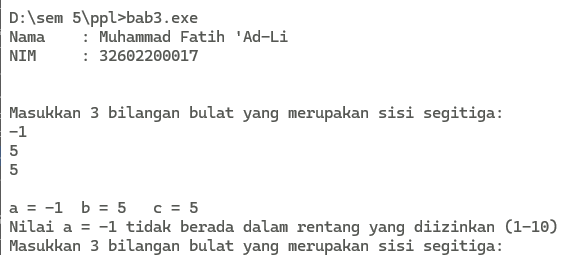
Gambar 3. 2 Output Test Case 2 Black Box Equivalence Class



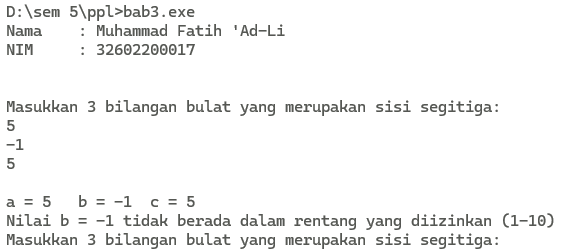
Gambar 3. 3 Output Test Case 3 Black Box Equivalence Class



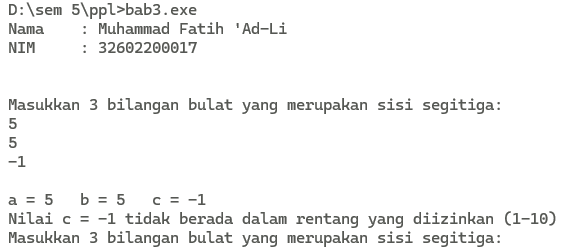
Gambar 3. 4 Output Test Case 4 Black Box Equivalence Class



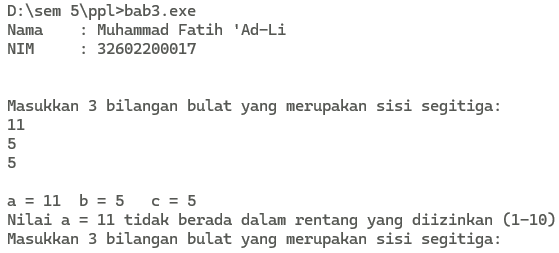
Gambar 3. 5 Output Test Case 5 Black Box Equivalence Class



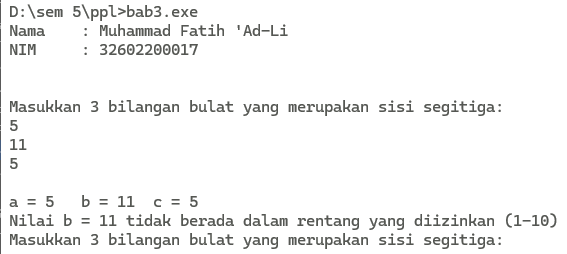
Gambar 3. 6 Output Test Case 6 Black Box Equivalence Class



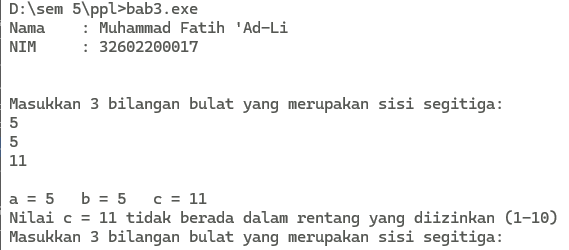
Gambar 3. 7 Output Test Case 7 Black Box Equivalence Class



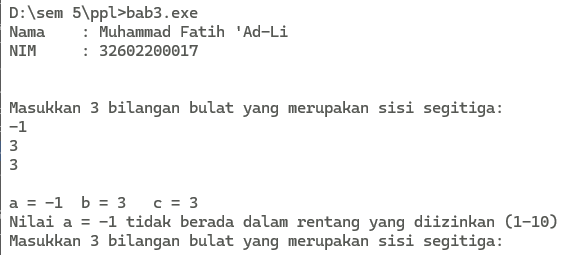
Gambar 3. 8 Output Test Case 8 Black Box Equivalence Class



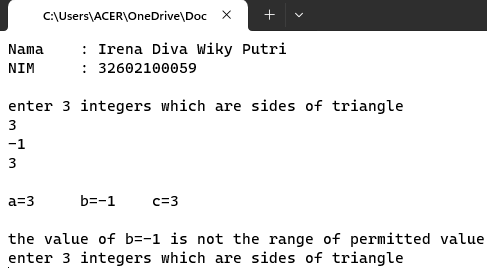
Gambar 3. 9 Output Test Case 9 Black Box Equivalence Class



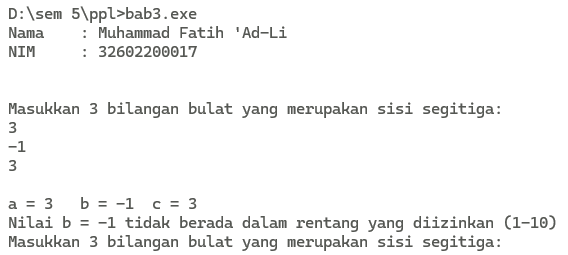
Gambar 3. 10 Output Test Case 10 Black Box Equivalence Class



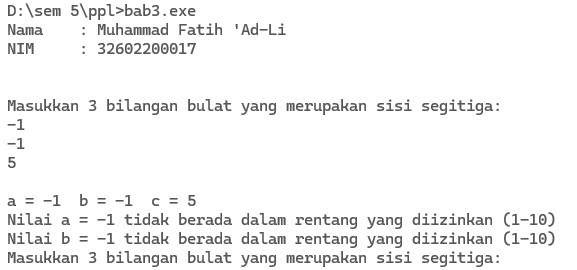
Gambar 3. 11 Output Test Case 11 Black Box Equivalence Class



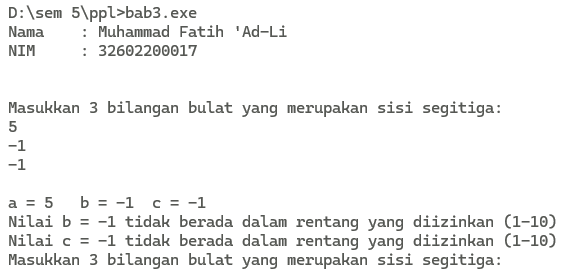
Gambar 3. 12 Output Test Case 12 Black Box Equivalence Class



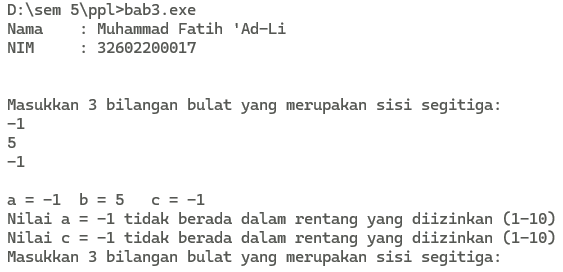
Gambar 3. 13 Output Test Case 13 Black Box Equivalence Class



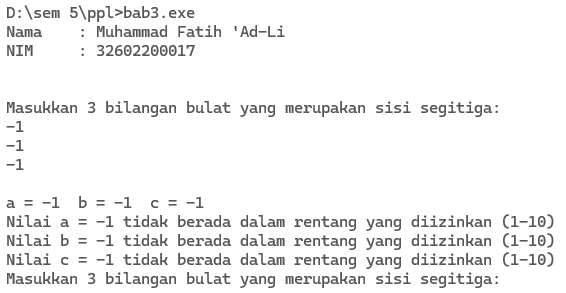
Gambar 3. 14 Output Test Case 14 Black Box Equivalence Class



Gambar 3. 15 Output Test Case 15 Black Box Equivalence Class



Gambar 3. 16 Output Test Case 16 Black Box Equivalence Class



Gambar 3. 17 Output Test Case 17 Black Box Equivalence Class

1. Penjelasan

Pengujian perangkat lunak dengan metode pengujian *black box* kelas ekuivalensi, yang membagi *input* domain yang berasal dari program kemudian dibagikan kedalam kelas-kelas data, yang mana kasus uji yang nantinya akan diujikan dapat diturunkan. Disamping itu, metode ini didasarkan kepada evaluasi persamaan yang ada pada kelas dari kondisi masukannya. Untuk itu, dalam hasil praktikum ini *output* yang ditampilkan seperti yang ditunjukkan pada gambar dapat ditunjukkan pada gambar 3.1 sampai dengan gambar 3.17 dengan *source code* bahasa C yang dijalankan dengan aplikasi *CodeBlocks*. Sehingga hasil pengujian yang didapatkan, pembagian partisi-partisi kelasnya adalah sebagai berikut, gambar 3.1-3.4 menunjukkan *output* pengujian *black box* dengan kelas ekuivalensi lemah. Kemudian gambar 3.5-3.10 menunjukkan *output* pengujian kelas ekuivalensi kuat lemah, lalu pada gambar 3.11-3.17 menunjukkan *output* pengujian kelas ekuivalensi kuat.

1. **Analisa**

Dari hasil percobaan praktikum dengan pengujian *black box* yang  
lebih berfokus ke dalam fungsionalnya dan menguji apakah *output* yang  
dikeluarkan sesuai dengan yang diinginkan atau tidak. di dalam percobaan  
bab 3 ini memiliki variabel normal yang bernilai 1-10 jika variabel yang  
dimasukkan melebihi atau kurang dari variabel normal maka program akan  
menampilkan *action error* dan program menampilkan *output the range* yang  
berarti melebihi dari nilai variabel normal, maka didapatkan analisanya sebagai berikut :

1. Analisa Pengujian *Case* 1

Pada pengujian *case* 1 pada gambar 3.1, ditampilkan variabel a = 5, b = 5, c = 5. Nilai tersebut masih dalam *range* yang telah ditetapkan dan  
dikarenakan nilai tersebut sama semua, maka ditampilkan *output* yang  
berupa jenis segitiga *equilateral triangle*, menunjukkan bahwa kondisi tersebut membentuk segitiga sama sisi.

1. Analisa Pengujian *Case* 2

Pada pengujian *case* ke- 2 pada gambar 3.2, ditampilkan variabel a = 2, b = 2, c = 3. Nilai ini masih dalam range yang telah ditentukan dan akan menampilkan *output isosceles triangle*, menunjukkan bahwa kondisi tersebut membentuk segitiga sama kaki.

1. Analisa Pengujian *Case* 3

Pada pengujian *case* ke- 3 pada gambar 3.3, ditampilkan variabel a = 3, b = 4, c= 5. Nilai ini masih dalam *range* yang telah ditentukan pada program dan *output* yang ditampilkan adalah *scalene triangle*, menunjukkan bahwa kondisi tersebut membentuk segitiga yang tidak sama semua sisinya.

1. Analisa Pengujian *Case* 4

Pada pengujian *case* ke-4 pada gambar 3.4, ditampilkan nilai variabel a = 4, b = 1, c = 2. Nilai ini masih dalam *range*/jangkauan yang telah ditentukan pada program, namun karena salah satu nilai terlampau tinggi dan tidak membentuk segitiga maka *output* yang keluar adalah *Not a Ttriangle*, menunjukkan bahwa kondisi tersebut bukan termasuk segitiga.

1. Analisa Pengujian *Case* 5

Pada pengujian *case* ke-5 pada gambar 3.5, ditampilkan nilai variabel a = -1, b = 5, c = 5. Dikarenakan salah satu nilai tidak memenuhi *range*/jangkauan pada program, maka *output* yang ditampilkan adalah *the value of* a = -1 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 6

Pada pengujian *case* ke-6 pada gambar 3.6, ditampilkan nilai variabel a = 5, b = -1, c = 5, dikarenakan salah satu nilai ada yang diluar jangkauan yaitu -1, dan tidak terbentuk segitiga maka *output* yang dikeluarkan adalah *the value of* b = -1 *is not the range of permitted value.*

1. Analisa Pengujian *Case* 7

Pada pengujian *case* ke-7 pada gambar 3.7, ditampilkan nilai variabel a = 5, b = 5, c = -1, dikarenakan salah satu nilai ada yang diluar jangkauan yaitu -1 maka tidak terbentuk segitiga dan *output* yang dikeluarkan adalah *the value of* c = -1 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 8

Pada pengujian *case* ke-8 pada gambar 3.8, ditampilkan nilai variable a = 11, b = 5, c = 5, pada kasus terdapat satu variabel yang tidak memnuhi jangkauan/ *range* yaitu 11, karena pada program hanya menampung angka yang rangenya <= 10. Maka, *output* yang dikeluarkan adalah *the value of* a =11 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 9

Pada pengujian *case* ke- 9 pada gambar 3.9, ditampilkan nilai variable a = 5, b = 11, c = 5, pada kasus ini terdapat satu nilai *variable* yang melebihi *range* yang telah ditetapkan pada program, yaitu 11, karena pada program hanya dapat menampung nilai hingga <= 10. Maka *output* yang ditampilkan adalah *the value of* b = 11 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 10

Pada pengujian *case* ke- 10 pada gambar 3.10, ditampilkan nilai variabel a = 5, b = 5, c = 11, pada kasus ini terdapat salah satu nilai *variable* yang melebihi *range*/ jangkauan yang telah ditetapkan pada program, yaitu 11, karena pada program hanya dapat menampung nilai hingga <= 10, maka *output* yang dihasilkan adalah *the value of* c = 11 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 11

Pada pengujian *case* ke- 11 pada gambar 3.11 ditampilkan nilai variabel a = -1, b = 3, c = 3, dalam kasus ini terdapat satu variabel yang tidak memenuhi *range*/ jangkauan yang telah ditetapkan pada program yaitu -1. Pada program nilai variabel >= 1. Maka *output* yang ditampilkan adalah *the value of* a = -1 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 12

Pada pengujian *case* ke- 12 pada gambar 3.12 ditampilkan nilai variabel a = 3, b = -1, c =3, dalam kasus ini terdapat satu *variable* yang tidak memenuhi *range*/ jangkauan yang telah ditetapkan pada program, yaitu -1. Pada program nilai variabel harus >= 1, maka *output* yang ditampilkan adalah *the value of* b = -1 *is not the range of permitted value.*

1. Analisa Pengujian *Case* 13

Pada pengujian *case* ke- 13 pada gambar 3.13, nilai yang ditampilkan adalah a = 3, b = 3, c = -1, dalam kasus ini salah satu variabel tidak memenuhi *range*/ jangkauan yang telah ditetapkan pada program, yaitu -1. Pada program nilai varibel harus >= 1. Maka *output* yang ditampilkan *the value of* c = -1 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 14

Pada pengujian *case* ke- 14 pada gambar 3.14 ditampilkan nilai variable a = -1, b=-1 , c = 5, pada kasus ini terdapat 2 *variable* yang tidak memenuhi *range*/ jangkauan yaitu a = -1, dan b = -1.pada program nilai variable harus >= 1. Maka *output* yang ditampilkan adalah *the value of* a = -1 *is not the range of permitted value dan the value of* b = -1 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 15

Pada pengujian *case* ke- 15 pada gambar 3.15, ditampilkan nilai variable a = 5, b = -1, c = -1, pada kasus ini terdapat 2 *variable* yang tidak memenuhi *range*/ jangkauan yaitu b = -1, dan c = -1. Pada program nilai variable harus >= 1. Maka *output* yang ditampilkan adalah *the value of* b = -1 *is not the range of permitted value dan the value of c = -1 is not the range of permitted value*.

1. Analisa Pengujian *Case* 16

Pada pengujian *case* ke- 16 pada gambar 3.16, ditampilkan nilai *variable* a = -1, b=5, c = -1, pada kasus ini, terdapat 2 *variable* yang tidak memnuhi *range*/ jangkauan yaitu a = -1, dan c = -1. Pada program nilai variable harus >= 1. Maka *output* yang ditampilkan adalah *the value of* a = -1 *is not the range of permitted value* dan *the value of* c = -1 *is not the range of permitted value*.

1. Analisa Pengujian *Case* 17

Pada pengujian *case* ke- 17 pada gambar 3.17, ditampilkan nilai *variable* a = -1, b = -1, c=-1, kasus ini, terdapat 3 *variable* yang tidak memnuhi *range*/ jangkauan. pada program nilai variabel harus >= 1. Maka *output* yang ditampilkan adalah *the value of* a = -1 *is not the range of permitted value* dan *the value of* b = -1 *is not the range of permitted  
value* dan *the value of* c = -1 *is not the range of permitted value*.

1. **Kesimpulan**

Berdasarkan hasil praktikum bab III diatas, menjelaskan mengenai pengujian *black box* adalah pengujian yang didasarkan pada pegecekan atau  
pengujian terhadap detail perancangan dari perangkat lunak. Pengujian *black box* diterapkan pada akhir tahapan proses pengujian. Hal ini  
dikarenakan pengujian ini tidak mementingkan struktur kontrol tetapi lebih  
memfokuskan pada domain informasi.

*black box* sendiri mempunyai strategi sistem yaitu *equivalence class testing* yang meliputi beberapa nilai yaitu nilai minimum variabel *input*, nilai di atas nilai minimum, nilai normal, nilai di bawah nilai maksimum dan nilai  
maksimum. Jika nilai termasuk dalam jangkauan maka akan menampilkan  
jenis segitiganya, kemudian jika nilai yang dimasukkan kurang dari nilai  
minimum dan lebih dari nilai maksimum maka program akan mengeluarkan  
*action invalid* karena di luar *range*/jangkauan.

**BAB IV  
PENGUJIAN *BLACKBOX* (2):  
PENGUJIAN ANALISIS NILAI BATAS**

****

****

# 

# BAB IV PENGUJIAN *BLACKBOX* (2): PENGUJIAN ANALISIS NILAI BATAS

1. **Tujuan**

Adapun tujuan praktikum ini, adalah :

1. Praktikan dapat mempersiapkan tahapan pengujian.
2. Praktikan dapat melakukan pengujian perangkat lunak dengan teknik Analisis Nilai Batas.
3. **Alat dan Bahan**

Alat dan bahan yang dibutuhkan dalam modul bab 4 ini adalah sebuah komputer yang dilengkapi dengan “CodeBlock” IDE (*Integrated Development Environtmen*) yang memiliki *Compiler* C.

1. **Dasar Teori**
2. **Analisis Nilai Batas (*Boundary Value Analysis* atau BVA)**

Teknik analisis nilai batas ini dilakukan karena adanya fenomena  
bahwa kesalahan sering terjadi pada daerah batas dari suatu *input*. Teknik  
ini tidak hanya memperhatikan batas suatu nilai input tapi juga  
memperhatikan batas nilai *output*.

Untuk permasalahan yang tidak diketahui dengan jelas cenderung  
menimbulkan kesalahan pada domain outputnya. BVA merupakan pilihan  
*test case* yang mengerjakan nilai yang telah ditentukan, dengan teknik  
perancangan *test case* melengkapi *test case equivalence partitioning* yang  
fokusnya pada *domaininput*. BVA fokusnya pada *domain output*.

1. **Petunjuk Pengujian *Boundary Value Analysis* (BVA)**

Petunjuk pengujian BVA adalah sebagai berikut :

1. Jika kondisi *input* berupa *range* yang dibatasi nilai a dan b, *test case*harus dirancang dengan nilai a dan b.
2. Jika sebuah kondisi *input* ditentukan dengan sejumlah nilai, *test case* harus  
   dikembangkan dengan mengerjakan sampai batas maksimal nilai
3. tersebut. Sesuai petunjuk 1 dan 2 untuk kondisi *output* dirancang *test  
   case* sampai jumlah maksimal.
4. Untuk struktur data pada program harus dirancang sampai batas  
   kemampuan. memerlukan sebuah nilai khusus, satu  
   kelas persamaan dan dua kelas persamaan didefinisikan.
5. **Prosedur Praktikum**

Berikut ini merupakan kode program dalam bahasa C untuk masalah segitiga yaitu menerima tiga *integer* yang dianggap sebagai tiga sisi segitiga dan menentukan apakah ketiga nilai tersebut merepresentasikan segitiga sama sisi, segitiga sama kaki, segitiga yang tidak sama semua sisinya atau tidak membentuk segitiga.

1. Menurunkan *Test Case*-nya.
2. Menjalankan *Test Case* tersebut.
3. Menganalisis hasil pengujian tersebut.

Program segitiga

#include <stdio.h>

#include <stdlib.h>

int main()

{

printf("Nama\t: Muhammad Fatih ‘Ad-Li\n");

printf("NIM\t: 32602200017\n\n");

int a,b,c,c1,c2,c3;

char istriangle;

do

{

printf("\nenter 3 integers which are sides of

triangle\n");

scanf("%d%d%d",&a,&b,&c);

printf("\na=%d\tb=%d\tc=%d",a,b,c);

c1 = a>=1 && a<=10;

c2= b>=1 && b<=10;

c3= c>=1 && c<=10;

if (!c1)

printf("\nthe value of a=%d is not the

range of permitted value",a);

if (!c2)

printf("\nthe value of b=%d is not the

range of permitted value",b);

if (!c3)

printf("\nthe value of c=%d is not the

range of permitted value”,c);

}

while(!(c1 && c2 && c3));

// Mengecek segitiga atau tidak

if( a<b+c && b<a+c && c<a+b )

istriangle='y';

else

istriangle ='n';

if (istriangle=='y')

if ((a==b) && (b==c))

printf("\nEquilateral triangle\n");

else if ((a!=b) && (a!=c) && (b!=c))

printf("\nScalene triangle\n");

else

printf("\nIsosceles triangle\n");

else

printf("\nNot a triangle\n");

return 0;

}

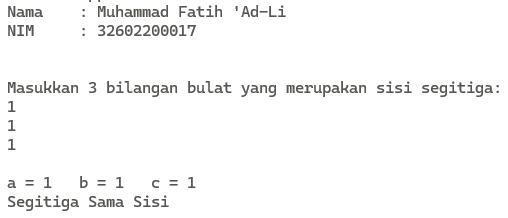
*Test Case*

1. *Test* data : Memasukkan 3 nilai *integer* (a, b, dan c)
2. *Pre-condition* : 1 ≤ a ≤ 10, 1 ≤ b ≤ 10 dan 1 ≤ c ≤ 10 dan a < b + c, b < a + c dan c < a + b
3. Deskripsi singkat : memeriksa apakah nilai merupakan segitiga sama sisi segitiga sama kaki, segitiga yang tidak sama semua sisinya atau tidak membentuk segitiga.

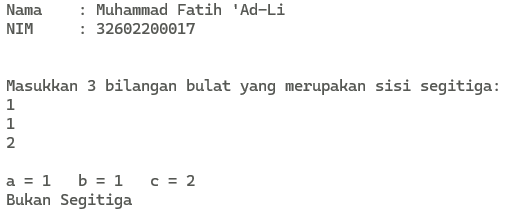
Table 4. 1 Test Case Boundary Value Analysis

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Case Id* | *Description* | *Input Data* | | | *Expected Output* | *Actual Output* | *Status* | *Comments* |
| a | b | c |
| 1 | *Enter the min value for* | 1 | 1 | 1 | *Should display the message equilateral triangle* | *Equila-teral triangle* | *Pass* |  |
| 2 | *Enter the min value for* 2 *itemsandmin* +1 *foranyoneitem1* | 1 | 1 | 2 | *Message should be displayed can’t form a triangle* | *Not a triangle* | *Pass* |  |
| 3 | *Enter the min value for* 2 *itemsandmin* +1 *foranyoneitem1* | 1 | 2 | 1 | *Message should be displayed can’t form a triangle* | *Not a triangle* | *Pass* |  |
| 4 | *Entertheminvaluefor* 2 *itemsandmin* +1 *foranyoneitem1* | 2 | 1 | 1 | *Message should be displayed can’t form a triangle* | *Not a triangle* | *Pass* |  |
| 5 | *Enterthenormalvaluefor* 2 *itemsand* 1 *itemisminvalue* | 5 | 5 | 1 | *Should display the message isosceles triangle* | *Iso-sceles triangle* | *Pass* |  |
| 6 | *Enterthenormalvaluefor* 2 *itemsand* 1 *itemisminvalue* | 5 | 1 | 5 | *Should display the message isosceles triangle* | *Iso-sceles triangle* | *Pass* |  |
| 7 | *Enter the normal value for* 2 *items and* 1 *item is min value* | 1 | 5 | 5 | *Should display the message isosceles triangle* | *Iso-sceles triangle* | *Pass* |  |
| 8 | *Enter the normal value for* a*,* b*, and* c | 5 | 5 | 5 | *Shoud display the message equilateral triangle* | *Equila-teral triangle* | *Pass* |  |

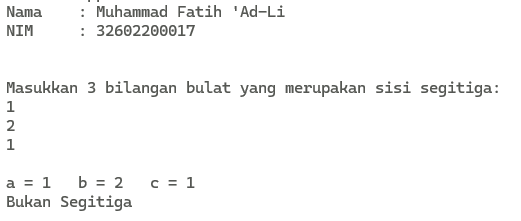
1. **Hasil Praktikum**
2. **Pengujian Sesuai Tabel Keputusan**
3. *Output*



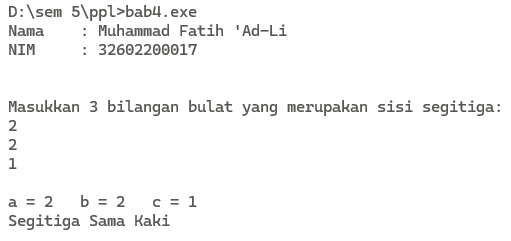
Gambar 4. 1 Output Test Case 1 Black Box Boundary Analysis Value



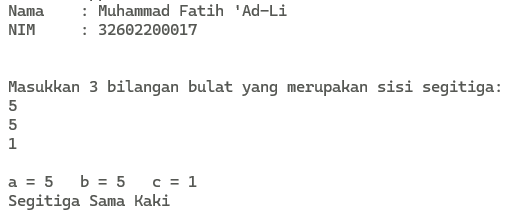
Gambar 4. 2 Output Test Case 2 Black Box Boundary Analysis Value



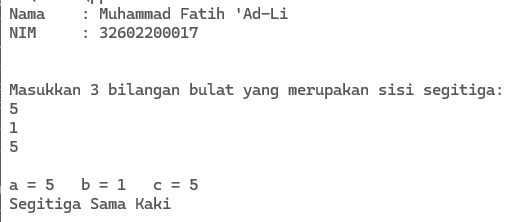
Gambar 4. 3 Output Test Case 3 Black Box Boundary Analysis Value



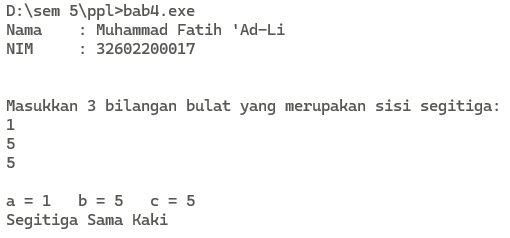
Gambar 4. 4 Output Test Case 4 Black Box Boundary Analysis Value



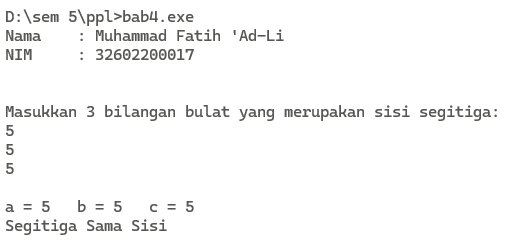
Gambar 4. 5 Output Test Case 5 Black Box Boundary Analysis Value



Gambar 4. 6 Output Test Case 6 Black Box Boundary Analysis Value



Gambar 4. 7 Output Test Case 7 Black Box Boundary Analysis Value



Gambar 4. 8 Output Test Case 8 Black Box Boundary Analysis Value

1. Penjelasan

Pengujian perangkat lunak dengan metode pengujian *black box* yang berfokus pada nilai batas atas, bawah, serta batas normal. Disamping itu, metode ini didasarkan kepada evaluasi persamaan yang ada pada kelas dari kondisi masukannya. Untuk itu, dalam hasil praktikum ini *output* yang ditampilkan seperti yang ditunjukkan pada gambar dapat ditunjukkan pada gambar 4.1 sampai dengan gambar 4.8 dengan *source code* bahasa C yang dijalankan dengan aplikasi *CodeBlocks*. Sehingga hasil pengujian yang didapatkan, pada batas-batasnya menunjukkan *output* berupa bangun segitiga yang terbentuk seperti, *equilateral triangle*, *isosceles triangle*, dan *not triangle* (bukan bangun segitiga).

1. **Analisa**

Dari hasil percobaan praktikum dengan teknik pengujian *black box* yang menggunakan nilai normal, pada batas atas, batas bawah dan batas normal dan terdapat beberapa *test case* dengan nilai minimum adalah bernilai 1, nilai normal adalah 2-9 dan untuk nilai maksimum adalah 10. dan akan di uji apakah akan terjadi invalid jika nilai yang di masukkan dibawah atau diatas nilai minimum dan maksimum.

1. Analisa Pengujian *Case* 1

Pada pengujian *case* 1 pada gambar 4.1, ditampilkan variabel a = 1, b = 1, c = 1, sesuai dengan aturan modul. Sehingga, nilai yang ditampilkan sesuai dengan jangkauan yang terdapat pada program dan semua sisinya memiliki panjang yang sama maka *output* yang ditampilkan adalah *equilateral triangle*, menunjukkan bahwa kondisi tersebut membentuk segitiga sama sisi.

1. Analisa Pengujian *Case* 2

Pada pengujian *case* ke- 2 pada gambar 4.2, ditampilkan variabel dengan nilai a = 1, b = 1, dan c = 2 sesuai dengan modul. Variabel tersebut sudah dapat memenuhi aturan yang terdapat pada program yaitu memenuhi jangkauan, namun nilai-nilai tersebut tidak memungkinkan untuk terbuat menjadi segitiga atau *not triangle*, hal ini karena tidak mungkin satu sisi sama panjang dengan jumlah dua sisi lainnya (1 dan 1).

1. Analisa Pengujian *Case* 3

Pada pengujian *case* ke- 3 pada gambar 4.3, ditampilkan variabel dengan nilai a = 1, b = 2, dan c = 1, sesuai dengan modul. Variabel tersebut sudah dapat memenuhi aturan yang terdapat pada program yaitu memenuhi jangkauan, namun nilai-nilai tersebut tidak memungkinkan untuk terbuat menjadi segitiga atau *not triangle*, hal ini karena tidak mungkin satu sisi sama panjang dengan jumlah dua sisi lainnya (1 dan 1).

1. Analisa Pengujian *Case* 4

Pada pengujian *case* ke-4 pada gambar 4.4, ditampilkan variabel dengan nilai a = 2, b = 1, dan c = 1, sesuai dengan modul. Variabel tersebut sudah dapat memenuhi aturan yang terdapat pada program yaitu memenuhi jangkauan, namun nilai-nilai tersebut tidak memungkinkan untuk terbuat menjadi segitiga atau *not triangle*, hal ini karena tidak mungkin satu sisi sama panjang dengan jumlah dua sisi lainnya (1 dan 1).

1. Analisa Pengujian *Case* 5

Pada pengujian *case* ke-5 pada gambar 4.5, ditampilkan variabel dengan nilai a = 5, b = 5, dan c = 1, sesuai dengan modul. Variabel tersebut sudah dapat memenuhi aturan yang terdapat pada program yaitu memenuhi jangkauan, dan karena 2 nilai variable sama, dan satu sisinya lebih kecil dari penjumlahan 2 sisi lain maka *output* yang ditampilkan  
*isosceles triangle* atau menunjukkan bahwa kondisi tersebut membentuk segitiga sama kaki.

1. Analisa Pengujian *Case* 6

Pada pengujian *case* ke-6 pada gambar 4.6, ditampilkan variabel dengan nilai a = 5, b = 1, dan c = 5, sesuai dengan modul. Variabel tersebut sudah dapat memenuhi aturan yang terdapat pada program yaitu memenuhi jangkauan, dan karena 2 nilai variabel sama, dan satu sisinya lebih kecil dari penjumlahan 2 sisi lain maka *output* yang ditampilkan  
*isosceles triangle* atau menunjukkan bahwa kondisi tersebut membentuk segitiga sama kaki.

1. Analisa Pengujian *Case* 7

Pada pengujian *case* ke-7 pada gambar 4.7, ditampilkan variabel dengan nilai a = 1, b = 5, dan c = 5, sesuai dengan modul. Variabel tersebut sudah dapat memenuhi aturan yang terdapat pada program yaitu memenuhi jangkauan, dan karena 2 nilai variable sama, dan satu sisinya lebih kecil dari penjumlahan 2 sisi lain maka *output* yang ditampilkan  
*isosceles triangle* atau menunjukkan bahwa kondisi tersebut membentuk segitiga sama kaki.

1. Analisa Pengujian *Case* 8

Pada pengujian *case* ke-8 pada gambar 4.8, ditampilkan variabel dengan nilai a = 5, b = 5, dan c = 5, sesuai dengan modul. Karena nilai yang ditampilkan sesuai dengan jangkauan yang terdapat pada program, dan semua sisinya memiliki panjang yang sama maka *output* yang ditampilkan adalah *equilateral triangle* menunjukkan bahwa kondisi tersebut membentuk segitiga sama sisi.

1. **Kesimpulan**

Berdasarkan hasil praktikum Bab IV diatas, menjelaskan mengenai pengujian *black box* metode kedua dengan teknik pengujian Nilai Batas atau bisa disebut *Boundary Value Analysis*. Teknik analisis nilai batas ini dilakukan karena adanya fenomena bahwa kesalahan sering terjadi pada daerah batas dari suatu *input*. Teknik ini tidak hanya memperhatikan batas suatu nilai input tapi juga memperhatikan batas nilai *output*.

Nilai Batas akan menguji program dengan memasukkan nilai-nilai disekitar nilai batas, sehingga akan menguji apakah program sesuai dengan kebutuhan sistem atau tidak. jika nilai yang dimasukkan berupa nilai normal, nilai minimum, dan nilai maksimum maka akan mengeluarkan *action* atau *output* membentuk salah satu jenis segitiga maupun tidak membentuk sebuah bangun segitiga.

**BAB V  
PENGUJIAN FUNGSIONAL DENGAN SELENIUM IDE**

****

****

# 

# BAB V PENGUJIAN FUNGSIONAL DENGAN SELENIUM IDE

1. **Tujuan**

Adapun tujuan dari praktikum ini adalah :

* + 1. Praktikan dapat memahami pendekatan pengujian terotomasi.
    2. Praktikan dapat menggunakan *Selenium* IDE untuk membuat *test suite* yang berisi minimal 2 *test case*.
    3. Praktikan dapat membuat *test suite* untuk dua *website*.

1. **Alat dan Bahan**

Adapun alat dan bahan pada praktikum ini adalah :

* + 1. Komputer *desktop*
    2. *Software Selenium* IDE
    3. Aplikasi *web*

1. **Dasar Teori**

Otomatisasi dimaksudkan sebagai otomatisasi untuk menghilangkan intervensi manusia, yaitu merupakan proses yang *self-controlling* atau *self- moving*. Perangkat lunak otomasi menawarkan *wizard* dan perintah otomatis selain mempunyai kemampuan merekam tugas dan *re-play*. Menggunakan program ini dapat melakukan rekaman tugas IT ataupun bisnis. Keuntungannya adalah cepat, handal, *repeatable*, *programmable*, *reusable*, dan lain-lain. (Jaya, 2019)

1. **Pengenalan *Selenium***

*Selenium* dibuat pada tahun 2004 oleh Jason R. Huggins dan timnya. Nama asalnya adalah *JavaScript Functional Tester* (JSFT). Merupakan *framework* pengujian berbasis *browser open source* yang awalnya dibangun oleh Thoughtworks, dengan fitur-fitur berikut:

* + - 1. 100% JavaScript dan HTML
      2. *Web testing tool*
      3. Mendukung pengujian aplikasi web 2.0
      4. Mendukung *cross-browser testing* (*on multiple browsers*)
      5. Mendukung berbagai sistem operasi
      6. *Cross-browser* – IE 6/7, Firefox 8+, Chrome. Opera, Safari 2.0+
      7. Pengujian dapat berjalan pada *browser* secara langsung
      8. Selenium dapat di-*deploy* pada Windows, Linux, dan Macintosh
      9. Diimplementasikan dengan teknologi *browser*
         1. JavaScript
         2. DHTML
         3. Frames

Komponen Selenium terdiri atas :

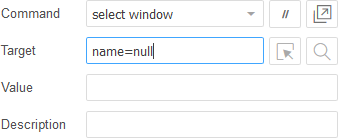
1. *Selenium* IDE
2. *Selenium Core*
3. *Selenium* RC
4. *Selenium GRID*
5. **Selenium IDE**

*Selenium* IDE (*Integrated Development Environment*) merupakan *tool* yang digunakan untuk membuat *Selenium* *test case*, dengan fitur sebagai berikut:

1. Merupakan *Chrome* *plugin*
2. Ekstensi *Chrome* yang memungkinkan paradigma *record*/ *play*
3. Otomasi perintah, tetapi asersi dimasukan secara manual
4. Membuat *locator* sesimpel mungkin
5. Berbasis *selences* (*set* perintah Selenium)

Selenium IDE terdiri dari :

1. Jendela *test case*
2. *Toolbar*
3. Menu bar
4. Jendela *log*/ *reference*/ UI-*Element*/ *Rollup*
   1. Jendela *test case* atau *test case pane* :
      * *Script* ditampilkan pada *test case pane*
      * Memiliki dua *tab*
      * Pertama untuk menampilkan perintah (*source code*)
      * Dan parameter dalam *readable* “*table*” format



Gambar 5. 1 Panel Selenium IDE

* 1. Toolbar

*Toolbar* berisi *buttons* untuk mengontrol eksekusi *test cases.*

* 1. Menu *bar*
     + Menu file

Menu file untuk membuat, membuka, dan menyimpan *test case* dan *test suite files*.

* + - Menu edit

Menu edit untuk operasi *copy*, *paste*, *delete*, *undo*, dan *select all* untuk mengedit perintah pada *test case*.

* + - Menu *option*

Menu *options* untuk mengubah *setting*, yaitu mengeset nilai *timeout value* untuk perintah tertentu, menambahkan ekstensi *user- defineduser* untuk *setting* dasar perintah Selenium dan menspesifikasi format atau bahasa yang digunakan ketika menyimpan *test cases*.

* 1. *Help bar*

Menjelaskan perintah *Selenium*. Set perintah tersebut disebut *selenese*. Perintah Selenium terdiri dari tiga jenis: Aksi, Asessori, dan Penegasan atau Aseri.

* + - Aksi

Yaitu Tindakan *user* pada aplikasi atau memerintahkan *browser* untuk melakukan sesuatu. Aksi merupakan perintah yang secara umum memanipulasi keadaan aplikasi.

1. Klik *link* **-** klik / *Click and wait*
2. Memilih item
   * + Asessori

Menguji keadaan aplikasi dan menyimpan hasilnya pada variabel, misal “*storeTitle*”.

* + - Asersi

Digunakan untuk memvalidasi aplikasi.

1. Untuk verifikasi halaman *web*
2. Untuk verifikasi teks
3. Untuk verifikasi *alerts*

Asersi dapat digunakan dalam mode :

1. *Assert*
2. *Verify*
3. *waitFor*

Contoh : "*assertText*","*verifyText*" dan "*waitForText*".

*Note* :

1. Ketika “*assert*” gagal, maka tes dihentikan.
2. Ketika “*verify*” gagal, maka tes tetap dilanjutkan
3. Perintah “*waitFor*” menunggu beberapa kondisi menjadi benar

Beberapa perintah Selenium yang sering digunakan untuk pengujian adalah sebagai berikut :

1. *Open* – membuka halaman menggunakan URL.
2. *Click* atau *click And Wait* melaksanakan operasi klik dan secara opsional menunggu halaman baru muncul.
3. *verifyTitle* atau *assertTitle* – memverifikasi judul halaman yang diharapkan.
4. *verifyTextPresent* – memverifikasi teks yang diharapkan muncul pada halaman *web.*
5. *verifyElementPresent* – memverifikasi elemen UI yang diharapkan, sebagaimana didefinisikan oleh HTML *tag* muncul pada halaman *web.*
6. *verifyText* – memverifikasi teks yang diharapkan HTML *tag* terkait pada halaman *web.*
7. *verifyTable –* memverifikasi konten tabel yang diharapkan.
8. *waitForPageToLoad* – menghentikan sementara eksekusi sampai halaman baru yang diharapkan muncul. Dipanggil secara otomatis ketika *clickAndWait* digunakan.
9. *waitForElementPresent* – menghentikan sementara eksekusi sampai elemen UI yang diharapkan muncul, sebagaimana didefinisika pada HTML tag.
10. **Mengeset *Recording* dan *Run***

Ketika Selenium IDE mula-mula dibuka, *record button* secara *default* akan *ON*. Selama perekaman, Selenium IDE secara otomatis akan menyisipkan perintah ke dalam *test case* berdasarkan tindakan yang dilakukan.

1. *Remember base* URL MODE

Menggunakan bais URL untuk menjalankan *test case* pada *domain* yang berbeda.

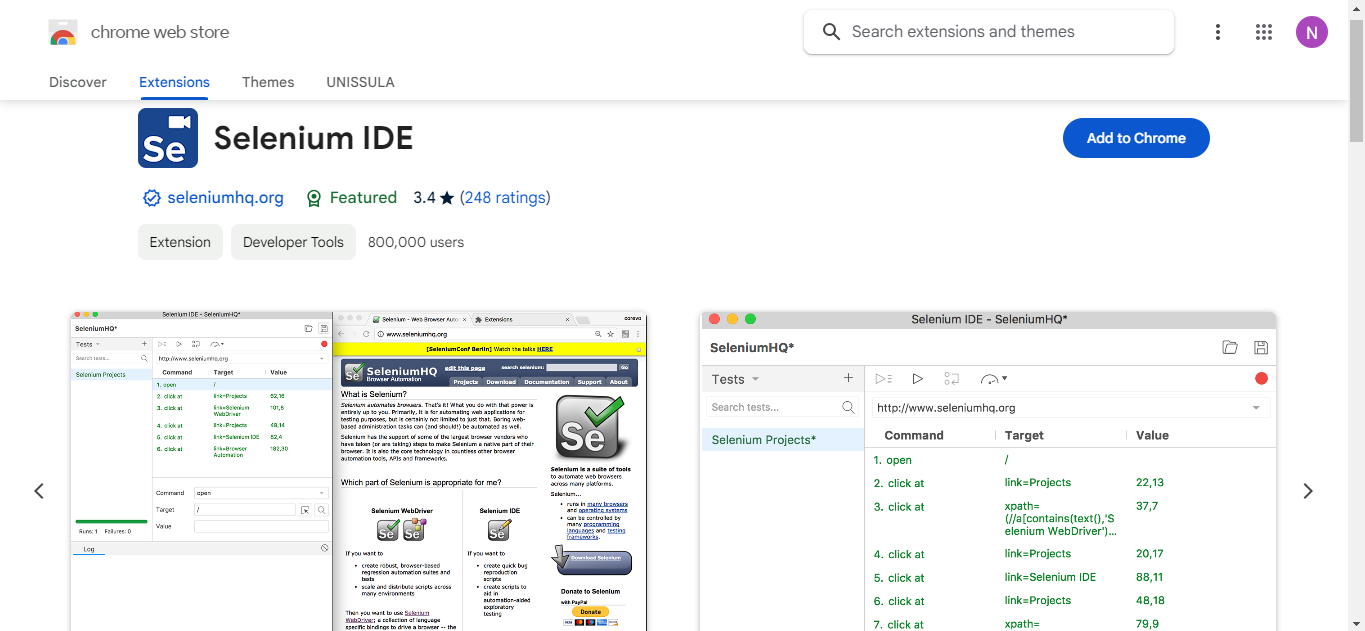
1. *Record absolute recording* MODE

Menjalankan *test case* pada *domain* tertentu.

1. **Menjalankan *Test Case***
   * + 1. Menjalankan *test case*: Klik *Run button* untuk menjalankan *test case* yang ditampilkan saat itu.
       2. Menjalankan *test suite*: Klik *Run All button* untuk menjalankan *test suite* yang di-*loading* saat itu.
       3. *Stop and start*: *Pause button* dapat digunakan untuk menghentikan *test case* ketika berjalan. *Icon button* ini akan berubah ke *Resume button*. Untuk melanjutkan klik *Resume*.
       4. *Stop in the middle*: *Breakpoint* pada *test case* dapat di-*set* yang menyebabkan berhenti pada perintah tertentu. Biasanya digunakan untuk melakukan *debugging* pada *test case*. Untuk mengeset suatu *breakpoint*, pilih perintah, klik kanan dan dari *context* menu, pilih *Toggle Breakpoint*.
       5. *Start from the middle*: Untuk memulai menjalankan perintah khusus pada tengah-tengah *test case*. Biasanya digunakan untuk melakukan *debugging* pada *test case*. Untuk mengeset *startpoint*, pilih *command*, *right-click*, dan dari *context* menu pilih *Set* atau *Clear Start Point*.
       6. *Run any single command*: *Double-click* suatu perintah tunggal untuk menjalankannya sendirian. Berguna ketika menulis perintah tunggal.
2. ***Test Suite***

Sebuah *test suite* adalah kumpulan tes. Seringkali kita akan menjalankan semua tes di *suite* tes sebagai salah satu *batch-job* kontinyu. Apabila menggunakan Selenium IDE, *test suite* juga dapat didefinisikan menggunakan file HTML sederhana. Sintaks juga sederhana. Tabel HTML mendefinisikan daftar tes dimana setiap baris mendefinisikan *path file system* untuk setiap tes.

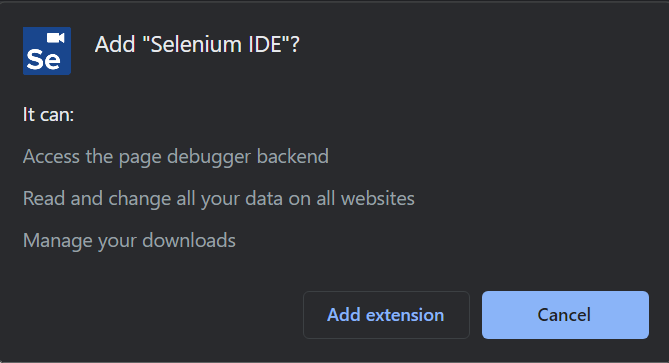
1. **Prosedur Praktikum**
   * + 1. Meng-*install Selenium* IDE
2. Men-*download* *Selenium* IDE dari *Google Chrome*



Gambar 5. 2 Website Selenium IDE

Gambar 5.2 adalah laman untuk men-*download extension Selenium* IDE. *Extension* tersebut dapat diunduh di *link* berikut: https://chrome.google.com/webstore/detail/selenium-[ide/mooikfkahbdckldjjndioackbalphokd](https://chrome.google.com/webstore/detail/selenium-ide/mooikfkahbdckldjjndioackbalphokd)

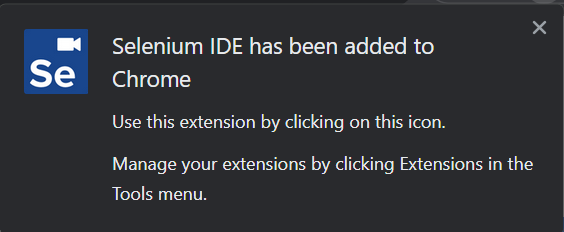
1. *Pop up* “*add extension*”



Gambar 5. 3 Pop up "add extension"

Gambar 5.3 adalah *pop up* “*add extension*”. Untuk melanjutkan proses *download* dan *install extension Selenium* IDE klik *button* “*add extension*”.

1. Instalasi selesai

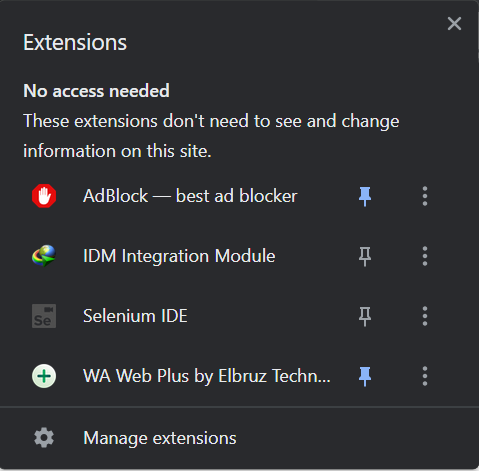


Gambar 5. 4 Berhasil menambahkan extension Selenium IDE pada Chrome

Gambar 5.4 adalah menandakan *extension Selenium* IDE berhasil ditambahkan pada *Chrome* dan siap untuk digunakan.

* + - 1. Membuka IDE

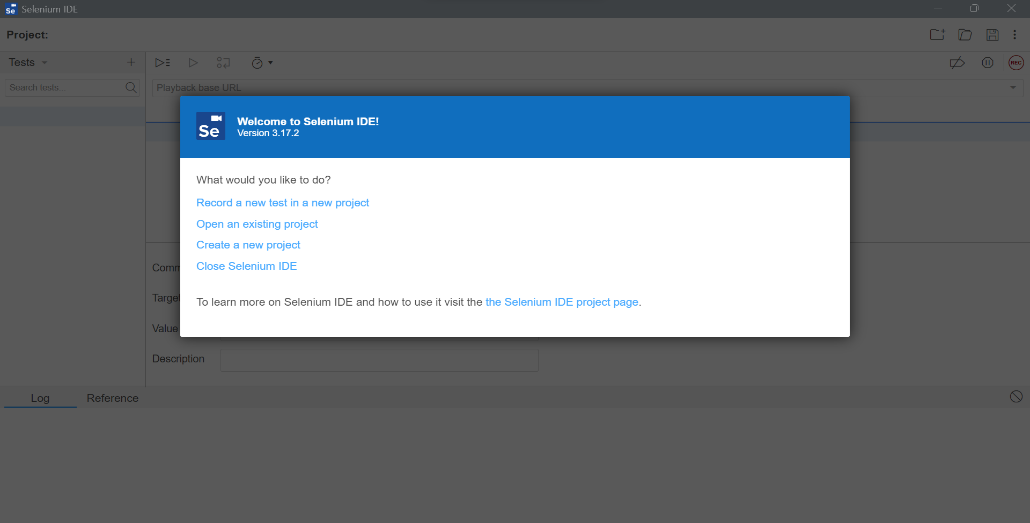
1. Klik *tab* “*extensions*”



Gambar 5. 5 Tab "extensions"

Gambar 5.5 adalah *tab* “*extensions*” yang ada pada *Google Chrome*. Posisi *tab* “*extensions*” berada di sebelah kiri *tab profile* email kita.

1. Klik *extensions* “*Selenium* IDE”



Gambar 5. 6 Halaman awal Selenium IDE

Setelah mengklik *extension* “*Selenium* IDE”, maka kita akan dibawa ke halaman awal dari *Selenium* IDE, seperti pada gambar 5.6. itu artinya kita sudah siap untuk membuat menjalankan *test suite* dan *test case* di *Selenium* IDE.

* + - 1. Menggunakan Selenium IDE, tulis *test suite* yang berisi minimal dua kasus uji

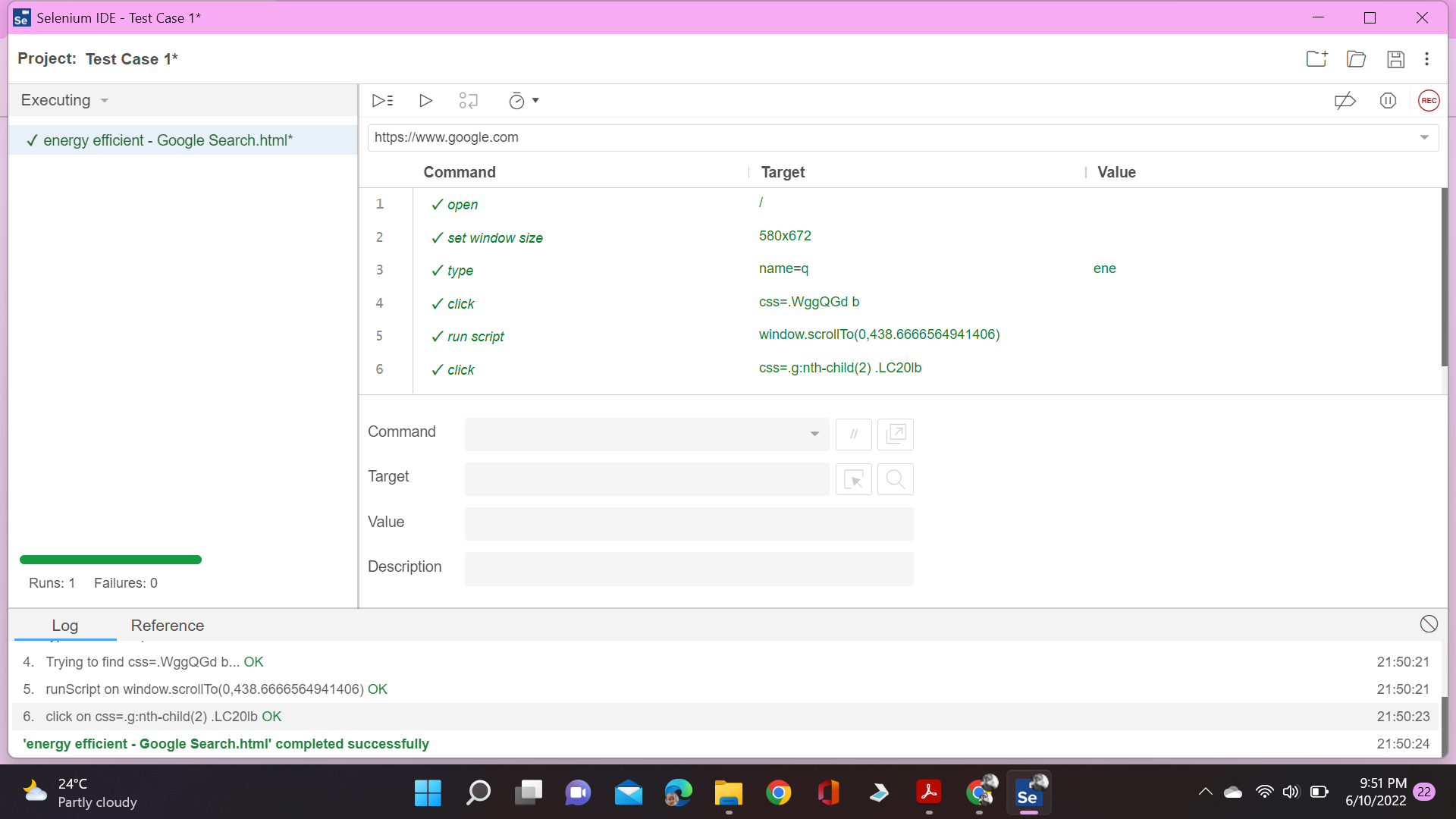
1. *Test case* 1
   * + - 1. Buka *Selenium* IDE, buat *test suite* dan *test case*
         2. Klik tombol “*recording*”
         3. Buka halaman *web* (misal: ketikan https://[www.google.com](http://www.google.com/))
         4. Ketikan “*energy efficient*” pada Google *search input box*
         5. Klik *search button*
         6. Verifikasi munculnya teks adalah “*energy efficient*”
         7. *Assert title* sebagai “*energy efficient* – *Google Search*”
         8. *Simpan* *test case* dengan ekstensi .html
2. *Test case* 2
3. Buka halaman *web* (misal: ketikan https://[www.google.com](http://www.google.com/))
4. Ketikan “Selenium RC” pada Google *search input box*
5. Klik *search button*
6. Verifikasi munculnya teks adalah “Selenium RC”
7. *Assert title* sebagai “*Selenium* RC – *Google Search*”
8. Simpan *test case* dengan ekstensi .html

Tahap – tahap *test suite*: Buat beberapa *test case* dan simpan setiap *test case* dengan ekstensi .html

1. Buka Google Chrome
2. Klik *tab* “*extensions*”
3. Klik *extension* “*Selenium* IDE”
4. Klik “*Create a new project*”
5. Tuliskan nama projeknya
6. Pilih *test suite*
7. Klik “*Add new suite*”
8. Klik “*Add test*”
9. Tambahkan beberapa *test cases*
10. Simpan *suite* dengan ekstensi .html
11. Jalankan *test suite*
    * + 1. Menjalankan *test suite* yang berisi dua *website* berbeda
12. *Test case* 1
    * + - 1. Buka *Selenium* IDE, buat *test suite* dan buat *test case*
          2. Klik tombol *recording*
          3. Buka halaman *web* (misal: https://[www.google.com](http://www.google.com/))
          4. Ketikan “*energy efficient*” pada Google *search input box*
          5. Klik *search button*
          6. Verifikasi munculnya teks adalah “*energy efficient*”
          7. *Assert title* sebagai “*energy efficient* – *Penelusuran Google*”
          8. Simpan *test case* dengan ekstensi .html
13. *Test case* 2
    * + - 1. Buka Chrome *web browser*
          2. Ketikan [https://www.yahoo.com](https://www.yahoo.com/) pada *address bar*
          3. Ketikan “*energy efficient*” pada Google *search input box*
          4. Klik *search button*
          5. Tunggu hasil pencarian sehingga menuju ke [https://search.yahoo.com](https://search.yahoo.com/)
          6. Verifikasi munculnya teks adalah “*energy efficient*”
          7. *Assert title* sebagai “*energy efficient* – *Yahoo Search Results*”
          8. Simpan *test case* dengan ekstensi .html

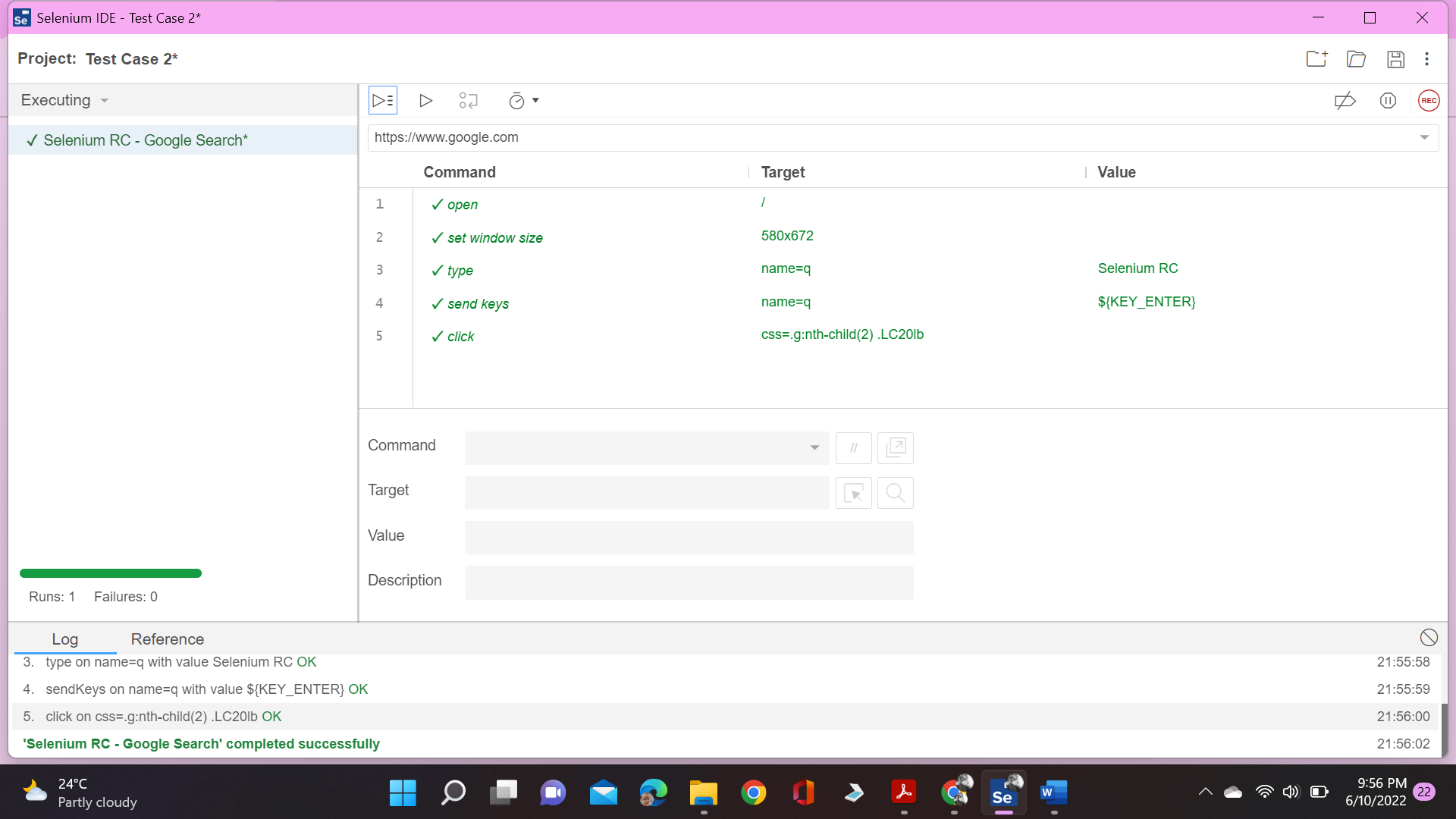
Tahap – tahap *test suite* : Buat beberapa *test case* dan simpan setiap *test case* dengan ekstensi .html.

1. Buka Google Chrome
2. Klik *tab* “*extensions*”
3. Klik *extension* “*Selenium* IDE”
4. Klik “*Create a new project*”
5. Tuliskan nama projeknya
6. Pilih *test suite*
7. Klik “*Add new suite*”
8. Klik “*Add test*”
9. Tambahkan beberapa *test cases*
10. Simpan *suite* dengan ekstensi .html
11. Jalankan *test suite*
12. **Hasil Praktikum**
    * 1. *Test suite* 1
13. *Test case* 1



Gambar 5. 7 Hasil Test Case "Energy Efficient - Google Search"

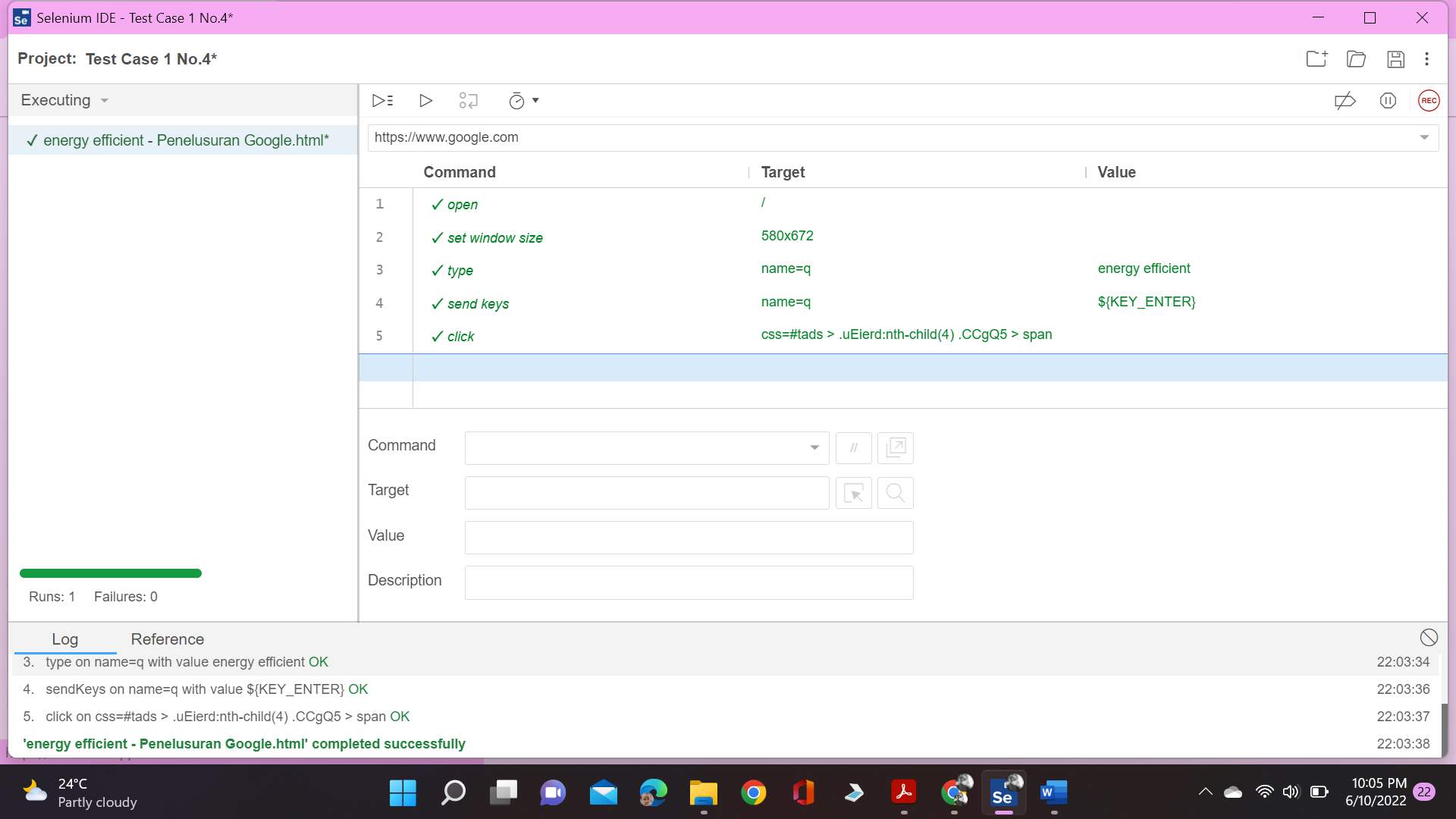
1. *Test case* 2



Gambar 5. 8 Hasil Test Case "Selenium RC - Google Search"

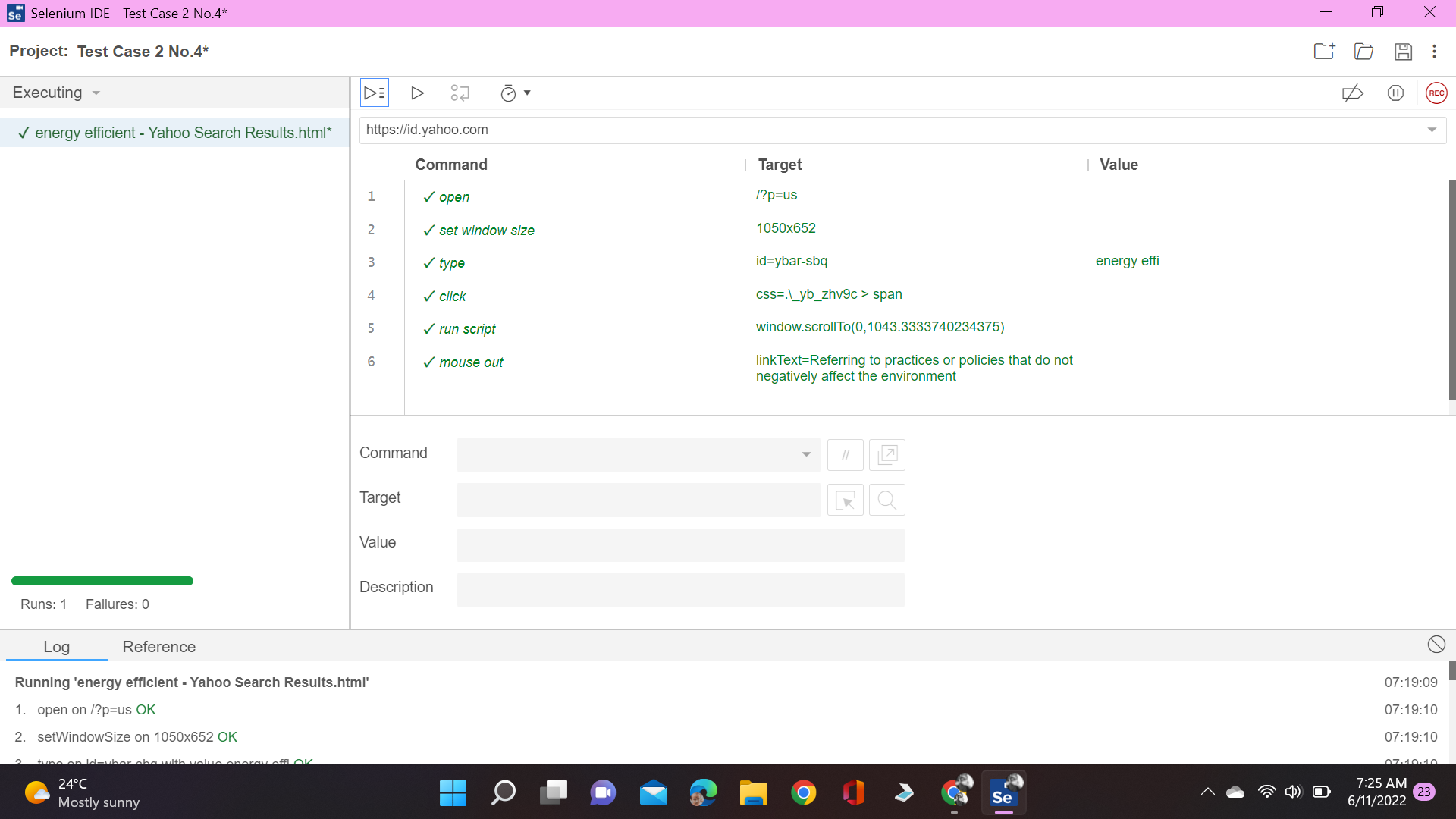
* + 1. *Test suite* “*energy efficient*”

1. *Test case* 1



Gambar 5. 9 Hasil Test Case "Energy Efficient - Penelusuran Google"

1. *Test case* 2



Gambar 5. 10 Hasil Test Case "Energy Efficient - Yahoo Search"

1. **Analisa**
   * 1. *Test suite* 1
   1. *Test case* 1

Pada *test suite* 1 *test case* ke-1 pada gambar 5.7 ditampilkan *command* pertama yaitu *open* yang artinya terjadi proses membuka jendela *browser* dengan *website google*. *Command* selanjutnya adalah *set window size*, yang artinya menentukan panjang dan lebar dari *tab window* yang digunakan. *Command* selanjutnya adalah *type,* yaitu proses untuk mengetikan kata untuk dicari pada *search engine*, pada kasus ini, kata yang diketikan adalah “*energy efficient*”. *Command* selanjutnya adalah *verify text* dan *asserts title*, untuk memverifikasi teks “*energy efficient*” dan *title* “*energy efficient – Penelusuran Google*” yang telah diketikan telah ditemukan atau tidak.

* 1. *Test case 2*

Pada *test suite* 1 *test case* ke-2 pada gambar 5.8 ditampilkan *command* pertama yaitu *open* yang artinya terjadi proses membuka jendela *browser* dengan *website google*. *Command* selanjutnya adalah *set window size*, yang artinya menentukan panjang dan lebar dari *tab window* yang digunakan. *Command* selanjutnya adalah *type,* yaitu proses untuk mengetikan kata untuk dicari pada *search engine*, pada kasus ini, kata yang diketikan adalah “Selenium RC”. *Command* selanjutnya adalah *verify text* dan *asserts title*, untuk memverifikasi teks “*Selenium RC*” dan *title* “*Selenium RC – Penelusuran Google*” yang telah diketikan telah ditemukan atau tidak.

* + 1. *Test suite 2*
    2. *Test case* 1

Pada *test suite* 2 *test case* ke-1 pada gambar 5.9 ditampilkan *command* pertama yaitu *open* yang artinya terjadi proses membuka jendela *browser* dengan *website google*. *Command* selanjutnya adalah *set window size*, yang artinya menentukan panjang dan lebar dari *tab window* yang digunakan. *Command* selanjutnya adalah *type,* yaitu proses untuk mengetikan kata untuk dicari pada *search engine*, pada kasus ini, kata yang diketikan adalah “*energy efficient*”. *Command* selanjutnya adalah *verify text* dan *asserts title*, untuk memverifikasi teks “*energy efficient*” dan *title* “*energy efficient* – *Penelusuran Google*” yang telah diketikan telah ditemukan atau tidak.

* + 1. *Test case 2*

Pada *test suite 2 test case ke-2* pada gambar 5.10 ditampilkan *command* pertama yaitu *open* yang artinya terjadi proses membuka jendela *browser* dengan *website yahoo*. *Command* selanjutnya adalah *set window size*, yang artinya menentukan panjang dan lebar dari *tab window* yang digunakan. *Command* selanjutnya adalah *type,* yaitu proses untuk mengetikan kata untuk dicari pada *search engine*, pada kasus ini, kata yang diketikan adalah “*energy efficient*”. *Command* selanjutnya adalah *verify text* dan *asserts title*, untuk memverifikasi teks “*energy efficient*” dan *title* “*energy efficient – Yahoo Search Results*” yang telah diketikan telah ditemukan atau tidak.

1. **Kesimpulan**

Pada pengujian BAB V merupakan pengujian fungsional dengan *selenium*. *Selenium* ini merupakan *framework* pengujian berbasis *browser open source* yang awalnya dibangun oleh Thoughtworks. Komponen yang terdapat pada selenium antara lain *Selenium* IDE, *Selenium Core*, *Selenium* RC, dan *Selenium* GRID. Pada bab ini menggunakan *Selenium* IDE (*Intergrated Development Environment*), merupakan *tool* yang digunakan untuk membuat *Selenium* *test case*. Fungsional *test* adalah sebuah proses *testing* dimana *software tester* berperilaku sebagai *end–user* dan memeriksa apakah fungsi dari sistem dapat berjalan dengan baik dari *user*. *Tools Selenium* IDE merupakan *integrated tool* untuk *agile testing*. Cara kerja *Selenium* IDE adalah dengan merekam semua aktifitas yang dilakukan *user* saat mengakses aplikasi berbasis web. Namun, *Selenium* IDE memiliki beberapa batasan seperti keterbatasan dalam pengelolaan variabel dan logika yang kompleks.

**BAB VI  
PENGUJIAN *STRESS* DENGAN *APACHE JMETER***

****

****

# BAB VI PENGUJIAN *STRESS* DENGAN *APACHE JMETER*

1. **Tujuan**

Adapun tujuan praktikum ini, adalah :

1. Praktikan dapat memahami pendekatan pengujian terotomasi.
2. Praktikan dapat menggunakan *Apache Jmeter* untuk menguji kinerja *stress* dari suatu *web server*.
3. **Dasar Teori**

*Stresstest* adalah jenis pengujian yang unik. Pengujian ini dilakukan dengan memberikan beban pada sebuah aplikasi untuk mengetahui titik performansi aplikasi tersebut. Dirancang untuk menghadapi situasi yang tidak normal pada saat program mengalami uji coba. *Stress* testing dilakukan oleh sistem untuk kondisi-kondisi seperti volume data yang tidak normal (melebihi atau kurang dari batasan) frekuensinya. *Stresstest* sering dilakukan pada aplikasi yang membutuhkan konkurasi maupun akses acak bersamaan dalam jumlah yang sangat banyak. Aplikasi dengan berbasis *web* dengan *request* yang sangat banyak menjadi contoh yang sangat menarik dalam hal ini. Dalam pengujian *stress test* akan menggunakan sebuah *tools Apache Jmeter*. *Apache Jmeter* Sebuah *Tool*/ Alat yang digunakan untuk melakukan *performancetest* pada sebuah *software*. *Apache Jmeter* dapat memberikan *request* dalam jumlah yang sangat banyak secara bersamaan dalam satu waktu pada *server Apache Jmeter*.

*JMeter* atau *Apache JMeter* merupakan *tool* pengujian kinerja yang bersifat *opensource* berbasis *Java*, sangat mudah digunakan dan mendukung beberapa bahasa pemrograman, termasuk HTTP / HTTPS, SOAP, JDBC, LDAP dan JMS. Bagi seorang QA Engineer JMeter bisa digunakan untuk melakukan *load*/ *StresstestingWebApplication*, FTP *Application* dan *Databaseservertest*. *JMeter* bisa dijalankan dengan 2 cara, dengan GUI atau

*non*-GUI (*Commandline*). Untuk *beginner* lebih baik menggunakan cara yang pertama. Mudah dan tanpa melakukan *scripting* tertentu. Tinggal membuat *testplan*, mengisikan berapa *thread* dan *sample* yang akan diujicobakan, *running* dan menganalisa hasil/ *report*.

*JMeter* dibangun untuk mendukung rencana uji/*testplan*. Dalam rencana uji terdapat *ThreadGroup*, *Samplers*/ pengendali, *Listeners*/ pendengar, *timer*, *Assertions*/ pernyataan, dan elemen lainnya. Setiap rencana uji adalah skenario pengujian kinerja dimana serangkaian Langkah *JMeter* akan dieksekusi rencana tersebut dijalankan. Tabel berikut memberikan gambaran singkat tentang elemen dasar yang dapat dimasukkan dalam rencana uji.

Table 6. 1 Deskripsi Apache JMeter

|  |  |
| --- | --- |
| **Elemen** | **Deskripsi** |
| *Thread Group* | Seperti halnya uji beban, eksekusi *multi*-*threaded*. *ThreadGroup* adalah apa yang mengontrol koneksi bersamaan untuk aplikasi Anda. |
| *Samplers* | Sebuah *sampler* adalah jenis dasar kontroler. Sangat sederhana, *samplers* memberitahu *JMeter* untuk mengirim permintaan (HTTP, SOAP, dll ...) ke *server*. |
| *Listeners* | *Listeners* adalah apa yang Anda gunakan untuk mengakses informasi yang dikumpulkan *JMeter* saat dijalankan. |
| *Timers* | *Timers* adalah bagaimana mengatur penundaan di *JMeter*. *timer* bekerja sebelum setiap permintaan/*request* yang dibuat oleh *thread* |
| *Assertions* | *Assertions*, seperti di alat uji, memungkinkan untuk memeriksa perilaku tertentu ketika mengeksekusi tes. *Assertion* menyediakan hasil standar lulus / gagal. |

1. **Alat dan Bahan**
2. Komputer *desktop*
3. *Software* Jmeter
4. *JVM* 7 atau lebih
5. *Web Server*
6. **Prosedur Praktikum**
7. Menginstal *Apache JMeter*
8. Instalasi *Java*

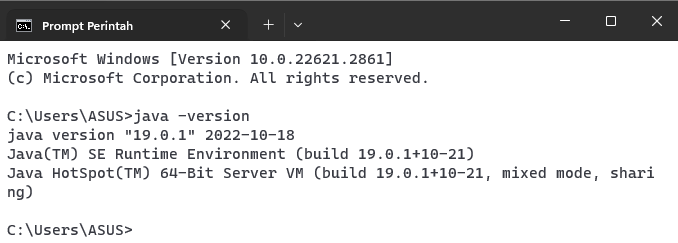
Karena *JMeter* adalah aplikasi berbasis Java, maka terlebih dahulu kita harus *install* Java dan memastikan jalan di sistem operasi kita. Untuk itu *download* dan *install* terlebih dahulu *Java* SE *Development Kit* di *link* URL berikut :

<http://www.oracle.com//technetwork/java/javase/downloads/jre8-downloads-2133155.html>.

Untuk cek apakah java sudah terinstall dengan baik, cek melalui *commandprompt* (Windows + R > cmd), ketikkan perintah :

Java -version

Jika sukses, maka akan muncul tampilan seperti di bawah :



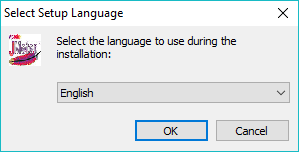
Gambar 6. 1 Pengecekan Java Version dengan CMD

1. Instalasi *JMeter*
2. Instalasi dari file *binary*

Paket-paket *JMeter* bisa di-*download* di situs <http://jmeter.apache.org/download_jmeter.cgi>. Pilih Binaries, *download* yang berekstensi \*.zip (misal: *apachejmeter*-3.0.zip). Tidak ada tahapan khusus dalam installasim *JMeter*, file zip yang telah berhasil di-*download*, tinggal ekstrak di *folder* yang diinginkan. Untuk keperluan *loadtesting*, kita masih membutuhkan beberapa *plugin* untuk ditambahkan ke *JMeter*. *Plugin* *JMeter* dapat dapat di-*download* di [https://jmeter-](https://jmeter-plugins.org/downloads/all/)*[plugin](https://jmeter-plugins.org/downloads/all/)*[s.org/downloads/all/](https://jmeter-plugins.org/downloads/all/). *Download* :

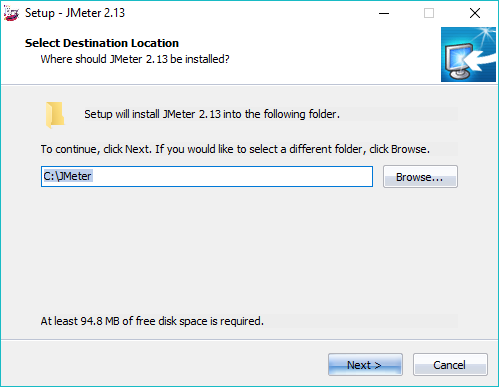
1. *JMeterPlugin*s-*Standard*. Cara menambahkan: ekstrak > kopikan seluruh isi lib\ext ke dalam *folder* lib\ext dari *Apache-Jmeter*.
2. *JMeterPlugin*s-*Extras*. Cara menambahkan: ekstrak > kopikan seluruh isi bin ke dalam *folder* bin apachejmeter, kopikan seluruh isi lib\ext\ ke dalam *folder* lib\ext\ dari apache-jmeter. Taruh *file*/ *folder* lain ke dalam *root Apache-Jmeter*.
3. *ServerAgent*. Ekstrak dan pisahkan dari direktori *JMeter* atau taruh di luar *folder JMeter*.
4. Lakukan hal yang sama untuk *plugin* lain.
5. Instalasi *Apache JMeter* *for Windows*

*Download* file *Apache JMeter* *for* Windows, buka lokasi penyimpanan file Jmeter, jika sudah diklik akan muncul *form* dialog pemilihan Bahasa disini kita pilih saja Bahasa inggris :

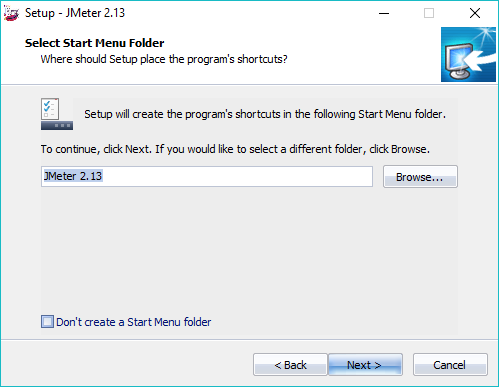


Gambar 6. 2 Pilih Pengaturan Bahasa

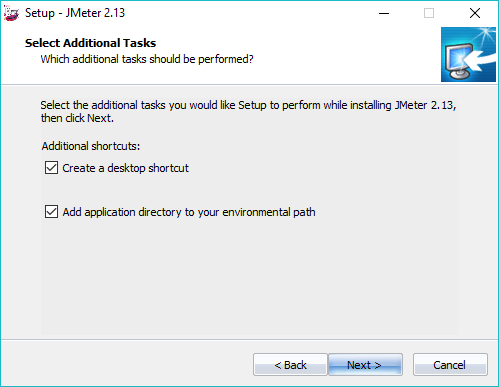
Setelah itu kita tinggal klik tombol *next* saja seperti gambar berikut:



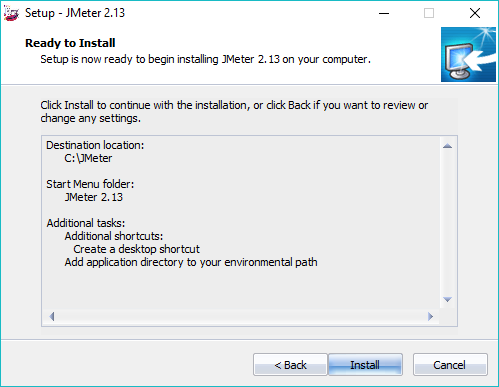
Gambar 6. 3 Memilih File Direktori Instalasi Apache JMeter



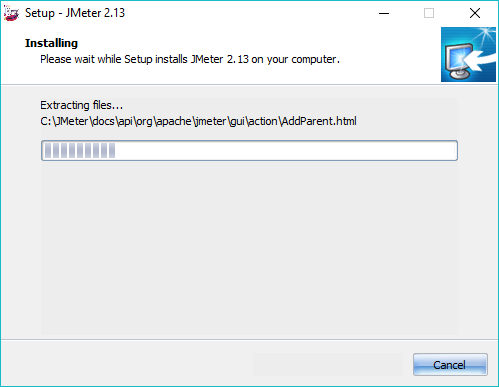
Gambar 6. 4 Start Menu Apache JMeter



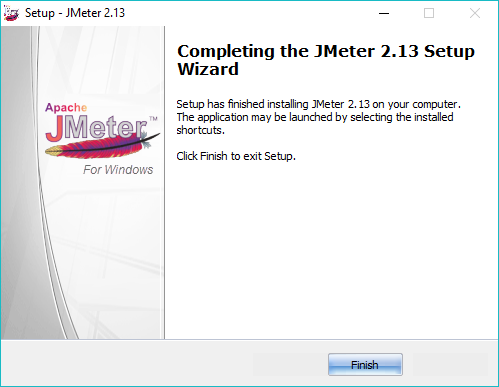
Gambar 6. 5 Menambahkan Shortcut Apache JMeter



Gambar 6. 6 Informasi Terkait Penginstalan

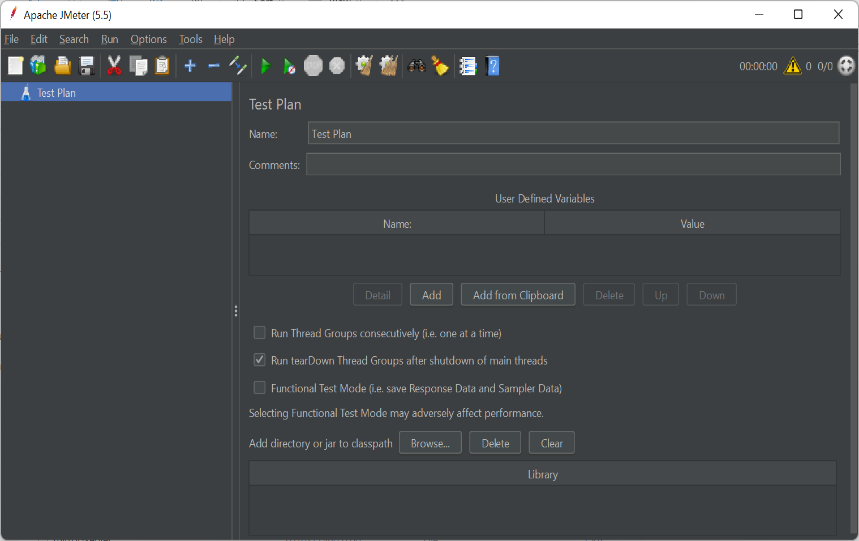


Gambar 6. 7 Proses Instalasi yang Sedang Berjalan



Gambar 6. 8 Instalasi Apache JMeter yang Telah Selesai

1. Menjalankan *Apache JMeter*
2. Cara menjalankan *Apache JMeter*, cukup masuk ke folder *bin* > *double click* *JMeter*.bat, dengan tampilan sebagai berikut.



Gambar 6. 9 Tampilan Awal Aplikasi JMeter

1. Untuk beberapa kasus tertentu (misalkan menjalankan *performserver*, diharuskan untuk menjalankan *ServerAgent* terlebih dahulu dengan cara *double* *click StartAgent*.bat). Alternatif kedua, mengklik ikon *JMeter*, akan muncul tampilan yang sama seperti gambar 6.9.
2. Membuat *Performance Testplan*

Setelah *JMeter* sudah berhasil dibuka, selanjutnya kita bisa menyiapkan *testplan*.

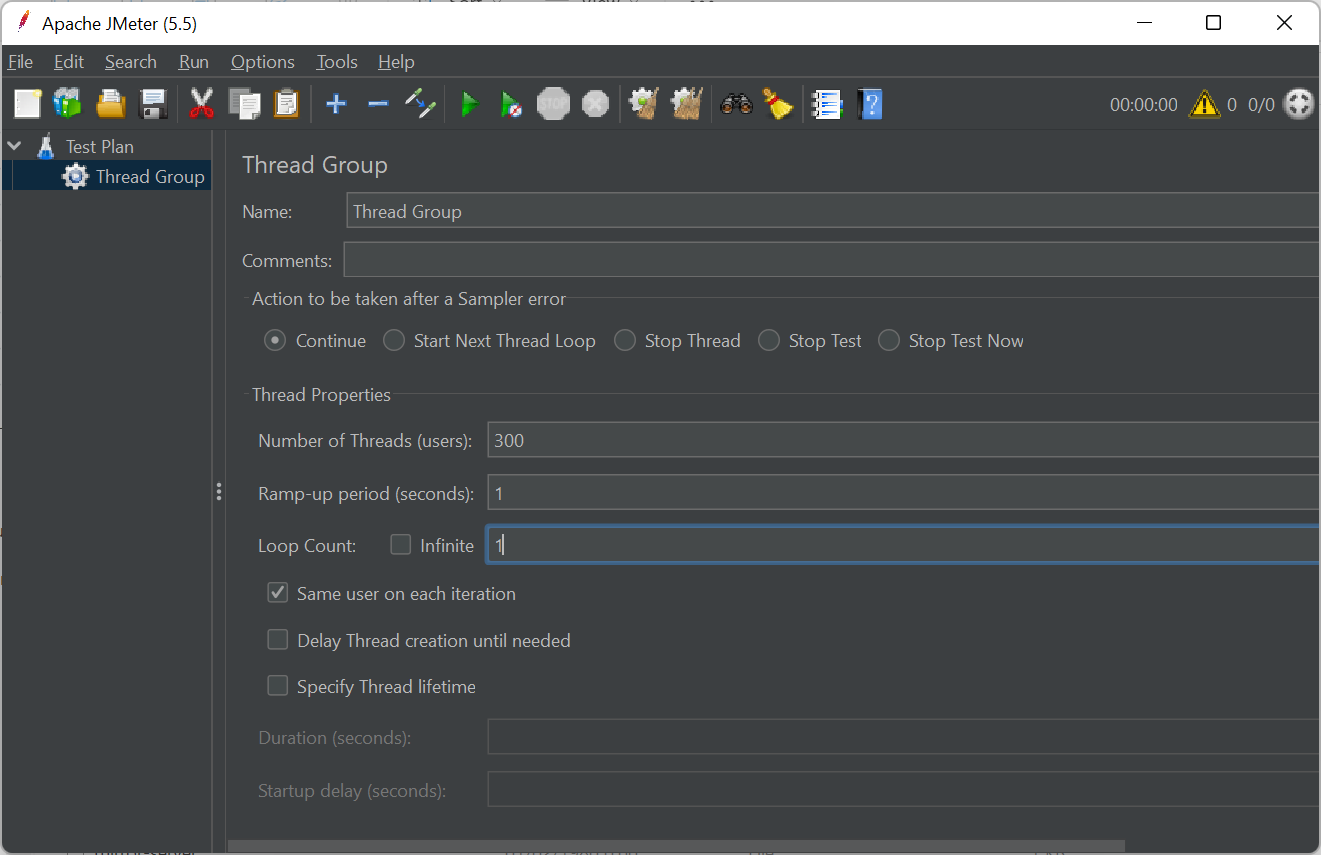
1. *Add Thread Group*. Menambahkan *traffic* atau *user* atau *visitor* kedalam komponen yang ingin di uji. Langkahnya sebagai berikut :
2. Klik kanan *testplan*.
3. *Add* - *Threads* (*Users*) – *ThreadGroup*
4. Dalam kontrol panel *ThreadGroup*, entri pada *ThreadProperties* :
5. *Numberofthreads* (*users*) : mengisi berapa *user* atau *visitor* yang akan mengakses *web*.
6. *Ramp*-*up period* (*in* *seconds*) : mengisi berapa waktu *delay* antara *user* satu dengan yang lainnya dalam mengakses web.
7. *Loopcount* : waktu eksekusi, bertahap atau seterusnya.
8. *Add* *JMeter* *Element* : menambahkan *webserver* atau IP *address* yang akan diuji. Caranya adalah :
9. Klik Kanan *ThreadsGroup*
10. *Add* - *ConfigElement* - HTTP *RequestDefaults*
11. Pada *WebServer* isi *ServerName* atau IP dan *Port*nya, atau gampangnya isi *website*/ url yang akan diuji. URL diisi dengan format<http://www>.
12. Jika tidak hanya halaman utama yang diuji, kita bisa menambahkan *path*/ *foldernya*, caranya :
13. Klik Kanan *ThreadsGroup*
14. *Add* > *Sampler* > HTTP *Request*
15. Isi *web server, port* dan *path*
16. *Add Listener*

Menampilkan proses dan hasil *test* secara grafis atau berbentuk tabel. Caranya adalah :

1. Klik kanan *TestPlan*
2. *Add* > *Listener* > *GraphResult*
3. *Add* > *Listener* > *ViewResultsinTable*
4. *Run Test*

Menjalankan *test* secara otomatis. Caranya adalah :

1. Simpan terlebih dahulu *TestPlan* yang telah kita buat di *File* > *Save* (Ctrl + S).
2. Klik *Run* atau Ctrl + R, JMeter akan mulai mensimulasikan sejumlah *user* dalam mengakses *webserver* yang telah ditentukan.
3. **Hasil Praktikum**
4. Pembuatan *Thread Group*
5. *Output*

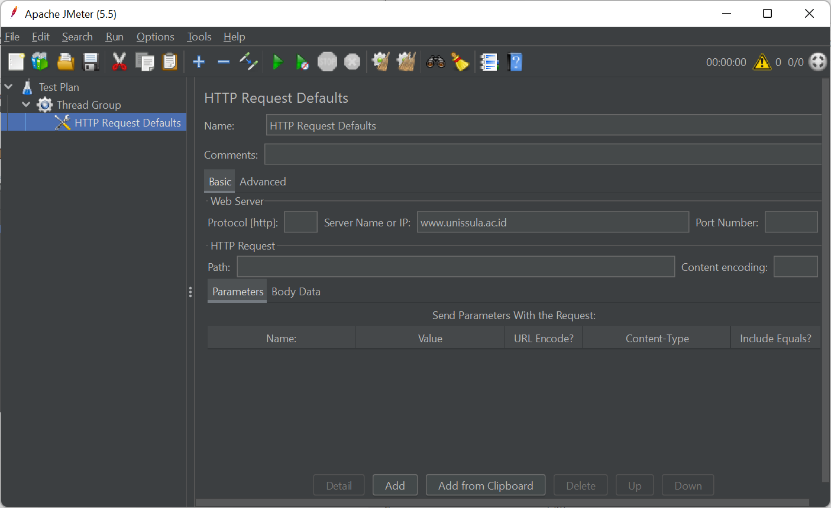


Gambar 6. 10 Menambahkan Thread Group Pada TestPlan

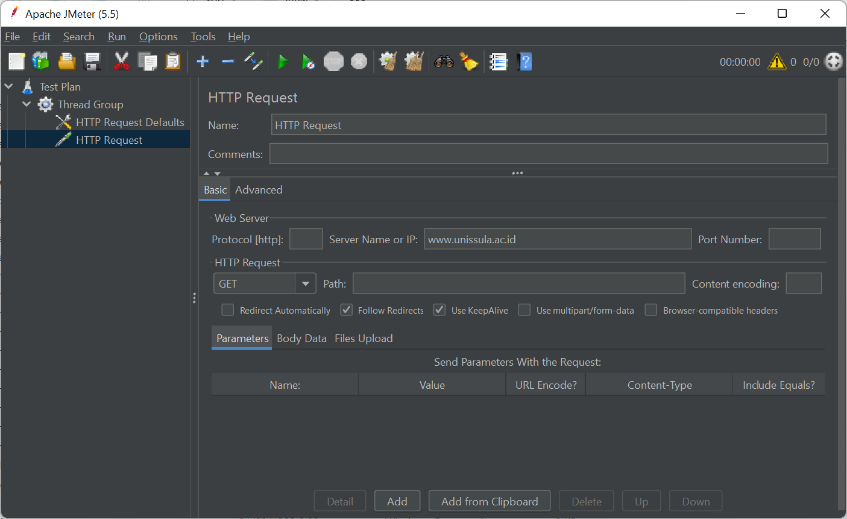
1. Penjelasan

Pembuatan *thread group* yang ditunjukkan pada gambar 6.10digunakan untuk menambahkan trafik dan kita meng*input*kan sejumlah 300 *user visitor* kedalam komponen yang akan diuji/ dijalankan.

1. Pembuatan *Element HTTP* Didalam *Threads Group*
2. *Output*



Gambar 6. 11 Membuat HTTP Request Default

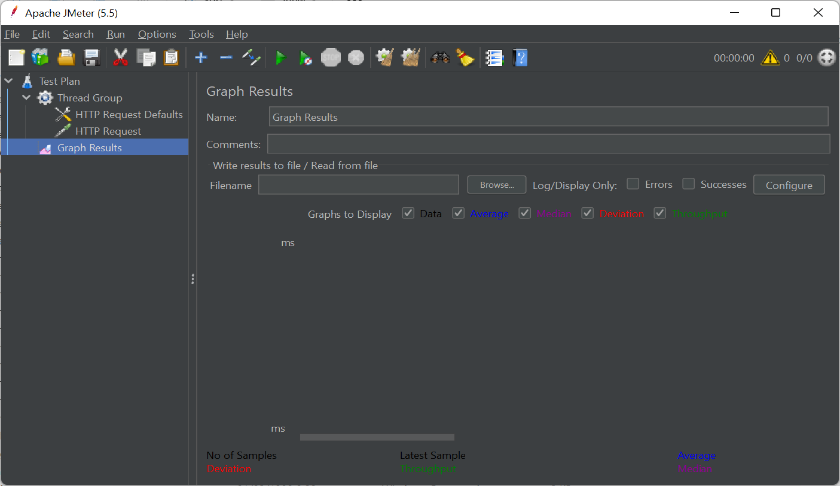


Gambar 6. 12 Penambahan HTTP Request

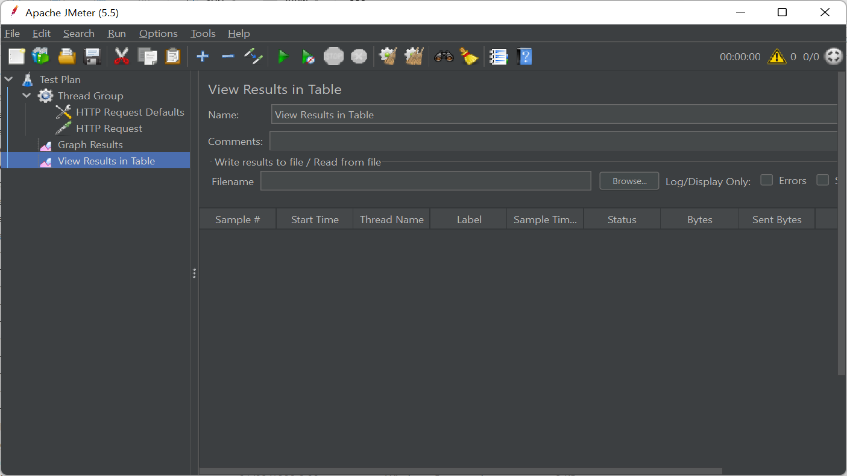
1. Penjelasan

Penambahan *element* seperti pada gambar 6.11 yang diambil dari *threads group* digunakan untuk menambahkan *web server* *name* atau dengan memasukkan alamat/ *url* *web* yang akan diujikan. Serta jika tidak hanya halaman utama yang *ditest*, kita dapat meminta atau *request* dengan ditambahkan foldernya seperti pada gambar 6.12.

1. Penambahan *Listener*
2. *Output*



Gambar 6. 13 Menambahkan Listener Graph Result

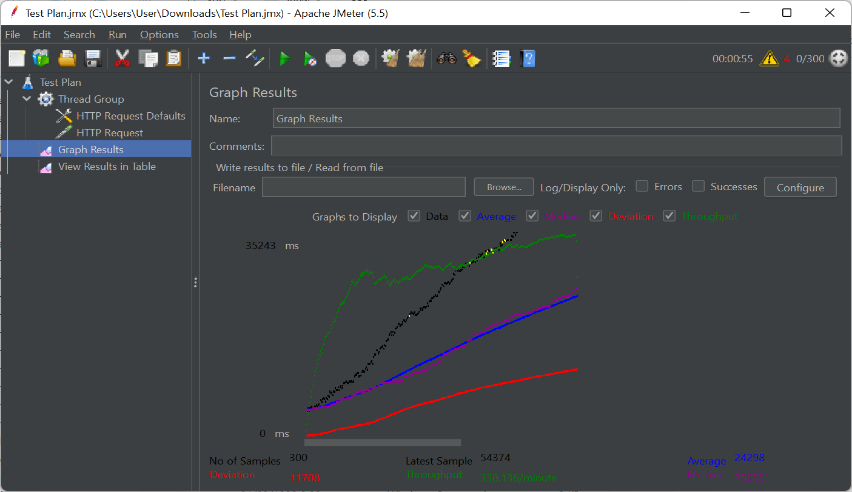


Gambar 6. 14 Menambahkan Listener View Result in Table

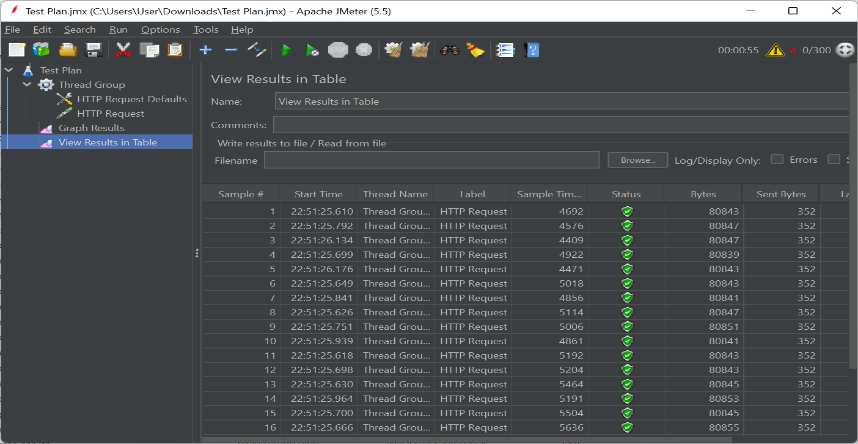
1. Penjelasan

Penambahan *listener* digunakan untuk menampilkan proses dan hasil uji yang ditampilkan dengan bentuk grafik yang disebut *graph result* seperti pada gambar 6.13, dan bentuk hasil lain yakni bentuk tabel disebut *view result in table* seperti ditunjukkan gambar 6.14.

1. Menjalankan *Test*
2. *Output*



Gambar 6. 15 Hasil Dalam Bentuk Grafik



Gambar 6. 16 Hasil Dalam Bentuk Tabel

1. Penjelasan

Setelah kita simpan t*test plan* yang sudah dibuat, maka kita dapat menjalankannya dengan memilih *option run* dengan ikon tombol *play* berwarna hijau, atau dapat pula dengan *shortcut* ctrl + R. Sehingga hasilnya adalah menampilkan hasil dengan bentuk grafik dan tabel seperti pada gambar 6.15 dan 6.16.

1. **Analisa**

Dari hasil percobaan praktikum diatas, maka pada *test* diatas ditampilkan beberapa pengecekan sebuah *website* dengan menggunakan *tool* *JMeter*. Dalam hal ini *website* yang di uji adalah [www.unissula.ac.id](http://www.unissula.ac.id). Sedangkan beberapa contoh yang diuji adalah data berupa IP *server* yang ada didalam *threadgroup*. Pada *test plan*, lama waktu respon, dan status. Sehingga didapatkan data diatas yang juga menampilkan hasil dimana terdapat 300 *user* *visitor* maka terdapat 1 *user* yang gagal mendapat respon atau tanggapan dari *website* unissula tersebut.

1. **Kesimpulan**

Berdasarkan hasil praktikum Bab VI, pengujian *stress* dengan menggunakan *tools Apache JMeter*. Pengujian *stress* dilakukan dengan memberikan beban pada sebuah aplikasi untuk mengetahui titik performansi aplikasi tersebut. Dirancang untuk menghadapi situasi yang tidak normal pada saat program mengalami uji coba. *JMeter* atau *Apache JMeter* merupakan *tool* pengujian kinerja yang bersifat *opensource* berbasis *Java*, sangat mudah digunakan dan mendukung beberapa bahasa pemrograman, termasuk HTTP / HTTPS, SOAP, JDBC, LDAP dan JMS. *JMeter* dibangun untuk mendukung rencana uji/*testplan*. Dalam rencana uji terdapat *ThreadGroup*, *Samplers*/ pengendali, *Listeners*/ pendengar, *timer*, *Assertions*/ pernyataan, dan elemen lainnya. Semakin banyak *user* yang mengakses *website* tersebut dan semakin banyak respon sukses yang diberikan maka *website* tersebut tahan ketika sudah di-*upload* dan digunakan secara global.

****

****

# DAFTAR PUSTAKA

Anjum, H., Imran, M., Jehanzeb, M., Khan, M., Chaudhry, S., Sultana, S., Shahid, Z., Zeshan, F., & Nazir, S. (2023). Otomatisasi Pengujian Aplikasi Web Toko Sembako Menggunakan Selenium IDE. LOGIC : Jurnal Ilmu Komputer Dan Pendidikan, 1(2), 303–309. https://journal.mediapublikasi.id/index.php/logic/article/view/1654

C Munaiseche, C. P., & Rorimpandey, G. C. (2021). Penerapan Metode Basis Path Analysis dalam Pengujian White Box Sistem Pakar. Sistem Informasi Dan Teknologi, 124–128.

Dace, A. A., Harya Daffa, M., Yeremias, Y., Sula, D., & Rahman, F. (2023). Pengujian Sistem Aplikasi Seleksi Sales Menggunakan Metode Black Box Teknik Equivalence Partitions. 1(3), 438–443. https://journal.mediapublikasi.id/index.php/logic

Debiyanti, D., Sutrisna, S., Budrio, B., Kamal, A. K., & Yulianti, Y. (2020). Pengujian Black Box pada Perangkat Lunak Sistem Penilaian Mahasiswa Menggunakan Teknik Boundary Value Analysis. Jurnal Informatika Universitas Pamulang, 5(2), 162. https://doi.org/10.32493/informatika.v5i2.5446

Muhammad Arofiq, N., Ferdo Erlangga, R., Irawan, A., & Saifudin, A. (2023). OKTAL : Jurnal Ilmu Komputer dan Science Pengujian Fungsional Aplikasi Inventory Barang Kedatangan Dengan Metode Black Box Testing Bagi Pemula. 2(5), 1322–1330. https://journal.mediapublikasi.id/index.php/oktal

Riswandi, Kasim, & Raharjo, M. F. (2020). Evaluasi Kinerja Web Server Apache menggunakan Protokol HTTP2. Journal of Engineering, Technology, and Applied Science, 2(1), 19–31. <https://doi.org/10.36079/lamintang.jetas-0201.92>

# LAMPIRAN

