TESIS – KI142502

STUDI PEMBENTUKAN KASUS UJI BERDASARKAN DIAGRAM ALIR DENGAN MENGHITUNG NILAI *NODE COVERAGE*, *TRANSITION COVERAGE*, DAN *SIMPLE PATH COVERAGE*

Desy Candra Novitasari
5115201020

DOSEN PEMBIMBING
Daniel Oranova, S.Kom. MSc. PD.Eng.
NIP. 19741123 20060410 01

Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.
NIP. 19751220 20011220 02

PROGRAM MAGISTER
BIDANG KEAHLIAN REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

[Halaman ini sengaja dikosongkan]

THESIS – KI142502

STUDY OF TEST CASE GENERATION BASED ON FLOWCHART WITH CALCULATE THE VALUE OF NODE COVERAGE, TRANSITION COVERAGE, AND SIMPLE PATH COVERAGE

DESY CANDRA NOVITASARI
5115201020

SUPERVISOR
Daniel Oranova, S.Kom. MSc. PD.Eng.
NIP. 19741123 20060410 01

Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.
NIP. 19751220 20011220 02

MASTER PROGRAM
DEPARTEMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

[Halaman ini sengaja dikosongkan]

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
DESY CANDRA NOVITASARI
Nrp. 5115201020

Dengan judul :
STUDI PEMBENTUKAN KASUS UJI BERDASARKAN DIAGRAM ALIR DENGAN MENGHITUNG NILAI NODE COVERAGE, TRANSITION COVERAGE DAN SIMPLE PATH COVERAGE

Tanggal Ujian : 12-7-2017
Periode Wisuda : 2016 Genap

Disetujui oleh:

Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng
NIP. 19741123006041001 (Pembimbing 1)

Dr.Eng. Chastine Fatichah, S.Kom., M.Kom.
NIP. 197512202001122002 (Pembimbing 2)

Dr. Ir. Siti Rochimah, M.T
NIP. 196810021994032001 (Penguji 1)

Sarwosri, S.Kom.,M.T.
NIP. 197608092001122001 (Penguji 2)

Rizky Januar Akbar, S.Kom, M.Eng
NIP. 198701032014041001 (Penguji 3)

Direktur Program Pasca Sarjana,

Dr. Agus Zainal Arifin, S.Kom.,M.Kom.
NIP. 197208091995121001

[Halaman ini sengaja dikosongkan]

STUDI PEMBENTUKAN KASUS UJI BERDASARKAN DIAGRAM ALIR DENGAN MENGHITUNG NILAI *NODE COVERAGE*, *TRANSITION COVERAGE*, DAN *SIMPLE PATH COVERAGE*

Nama Mahasiswa : Desy Candra Novitasari
NRP : 5115 201 020
Pembimbing : Daniel Oranova, S.Kom. M.Sc. PD.Eng.
Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

ABSTRAK

Pembentukan kasus uji merupakan salah satu langkah yang paling sulit pada tahap pengujian. Untuk merancang kasus uji dalam jumlah yang banyak dan menguji keseluruhan kasus uji tersebut, sangat dibutuhkan upaya yang cukup intensif dengan waktu yang lama. Pembentukan kasus uji yang dihasilkan secara otomatis dapat mengurangi biaya pengembangan perangkat lunak.

Model Based Testing merupakan salah satu pendekatan pengujian yang dapat dilakukan pada tahap awal siklus hidup perangkat lunak, yaitu pada fase desain. Banyak penelitian terkait pembentukan kasus uji yang hanya berfokus pada UML diagram. Padahal tidak semua perangkat lunak dirancang menggunakan UML diagram. Oleh karena itu, penelitian ini berfokus pada pembentukan kasus uji berdasarkan diagram alir (*flowchart*). Masukan yang digunakan dalam penelitian adalah diagram alir, dan keluaran yang dihasilkan adalah kasus uji. Diagram alir tersebut dirancang dari kode sumber dari program *console* sederhana menggunakan bahasa pemrograman C#.

Tahapan penelitian yang dilakukan adalah diagram alir dikonversi ke dalam bentuk *Flowchart Graph* (FG). Selanjutnya, dari FG dihasilkan banyak kasus uji (*test case*), karena sebuah FG dapat menghasilkan lebih dari satu kasus uji. Kasus uji dibentuk berdasarkan 3(tiga) kriteria cakupan yang sudah ditentukan, yaitu *Node Coverage, Transition Coverage*, dan *Simple Path Coverage*. Berdasarkan analisa pengujian terhadap 10 diagram alir, didapatkan hasil rata-rata *Node Coverage* 77,4%, *Transition Coverage* 64,7%, dan *Simple Path Coverage* 78,5%. Sedangkan akurasi yang didapatkan adalah 60%.

Kata Kunci: *software testing, test case pgeneration, flowchart model, C#, automatic test case*

[Halaman ini sengaja dikosongkan]

STUDY OF TEST CASE GENERATION BASED ON FLOWCHART WITH CALCULATE THE VALUE OF NODE COVERAGE, TRANSITION COVERAGE, AND SIMPLE PATH COVERAGE

Student Name : Desy Candra Novitasari
NRP : 5115 201 020
Supervisor : Daniel Oranova, S.Kom. M.Sc. PD.Eng.
Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

ABSTRACT

Generation of test cases is one of the most difficult steps in the testing phase. To design the test cases in large numbers and test all the test cases, it will be desperately needed effort intensive with long lead times. Automatic generation of test case can reduce software development costs.

Model Based Testing is a testing approach that can be done in the early stages of the software life cycle, namely in the design phase. Lot of research related to the generation of test cases that focus on UML diagrams. Whrereas, not all software is designed using UML diagrams. Therefore, this research focuses on the generation of test cases based on a flowchart. The input used in the study is a flow chart, and the resulting output is a test case. The flowchart is designed from the source code of a simple console program using the C# programming language.

The research step is the flowchart is converted into Flowchart Graph (FG). Next, the flowchart are converted to Flowchart Graph (FG). FG will produce a lot of test cases, because a FG can produce more than one test case. Test cases formed based on three criteria specified coverage, namely Node Coverage, Transition Coverage, and Simple Path Coverage. Based on the test analysis of 10 flowcharts, Node Coverage 77,4%, Transition Coverage 64,7%, dan Simple Path Coverage 78,5%..While the accuracy is 60%.

Keywords: *software testing, test case generation, flowchart model, C#, automatic test case.*

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tesis yang berjudul "Optimasi Kasus Uji Berdasarkan Diagram Alir Menggunakan Algoritma Genetika".

Pengerjaan Tesis ini merupakan suatu kesempatan yang sangat berharga bagi penulis untuk belajar memperdalam ilmu pengetahuan.Terselesaikannya buku Tesis ini, tidak terlepas dari bantuan dan dukungan semua pihak. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada:

1. Allah SWT atas limpahan rahmat-Nya sehingga penulis dapat menyelesaikan Tesis ini dengan sangat baik.
2. Bapak Sukamto dan Ibu Rini Utami,selaku orang tua penulis yang selalu mendoakan agar selalu diberikan kelancaran dan kemudahan dalam menyelesaikan Tesis ini. Serta menjadi motivasi terbesar untuk mendapatkan hasil yang terbaik.
3. Bapak Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng dan Ibu Dr. Eng. Chastine Fatichah, S.Kom., M.Kom., selaku dosen pembimbing yang telah memberikan kepercayaan, motivasi, bimbingan, nasehat, perhatian serta semua bantuan yang telah diberikan kepada penulis dalam menyelesaikan Tesis ini.
4. Dr. Ir. Siti Rochimah, M.T, Ibu Sarwosri,S.Kom., M.T, dan Bapak Rizky Januar Akbar, S.Kom, M.Eng. selaku dosen penguji yang telah memberikan bimbingan, saran, arahan, dan koreksi dalam pengerjaan Tesis ini.
5. Bapak Waskitho Wibisono, S.Kom., M.Eng., PhD selaku ketua program pascasarjana Teknik Informatika ITS, Bapak Prof. Dr. Ir. Joko Lianto M.Sc., selaku dosen wali penulis dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
6. Segenap staf Tata Usaha yang telah memberikan segala bantuan dan kemudahan kepada penulis selama menjalani kuliah di Teknik Informatika ITS.
7. Muh. Ngaliman, Diana Candra Novitasari serta seluruh keluarga besar yang selalu memberi semangat, doa, dukungan dan hiburan kepada penulis.
8. Teman senior Mbak Siska Arifiani dan Koko Felix Handani yang selalu menjadi teman diskusi yang baik, serta selalu mendo'akan dan mendukun penulis. Insya Allah rekan-rekan semua diberikan kemudahan dalam penyelesaian tesis.

9. Rekan-rekan angkatan 2015 Pasca Sarjana Teknik Informatika ITS yang telah menemani dan memberikan bantuan serta motivasi untuk segera menyelesaikan Tesis ini.
10. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu disini yang telah membantu terselesaikannya Tesis ini.

Sebagai manusia biasa, penulis menyadari bahwa Tesis ini masih jauh dari kesempurnaan dan memiliki banyak kekurangan.Sehingga dengan segala kerendahan hati, penulis mengharapkan saran dan kritik yang membangun dari pembaca.

Surabaya, Juli 2017

DAFTAR ISI

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Pengujian perangkat lunak adalah proses verifikasi yang menjanjikan jaminan kualitas perangkat lunak, bahwa perangkat lunak yang dikembangkan dapat memenuhi kebutuhan pelanggan (Sumalatha, 2009). Pengujian memiliki peran yang sangat penting dalam menjamin dan mengontrol kualitas pada siklus hidup perangkat lunak serta keandalan perangkat lunak (Jena, Swain, & Mohapatra, 2014). Tujuan utama dari pengujian adalah mengurangi atau meminimalkan jumlah kasus uji untuk menghemat waktu dan sumber daya. Waktu yang dibutuhkan untuk pengujian bergantung pada besar dan kompleksnya perangkat lunak yang dikembangkan. Lebih dari 50 persen biaya dan waktu dihabiskan untuk melakukan pengujian pada pengembangan perangkat lunak (Verma & Arora, 2016).

Secara umum, teknik pengujian dapat dilakukan secara kotak putih (*white box*) dan kotak hitam (*black box*). Kotak putih merupakan sebuah teknik pengujian yang menggunakan *Software Under Test* (SUT) atau perangkat lunak yang diuji sebagai petunjuk pengujian yang akan dilakukan. Teknik pengujian kotak putih bergantung pada analisa dampak perubahan dari kode program. Sehingga teknik pengujian kotak putih dapat dikatakan sebagai teknik untuk memvalidasi perubahan kode yang terjadi, tidak untuk memvalidasi spesifikasi kebutuhan dari suatu perangkat lunak. Sedangkan teknik pengujian kotak hitam merupakan teknik pengujian yang menguji level yang lebih tinggi dari suatu perangkat lunak seperti desain tampilan, ataupun spesifikasi kebutuhan. (Jena et al., 2014)

Kasus uji digunakan untuk memanipulasi perangkat lunak dalam pemanfaatan sumber daya yang tepat dan meminimalkan biaya perangkat lunak serta mengurangi waktu pelaksanaan (Engineering, Tomar, & Singh, 2016). Pembentukan kasus uji merupakan salah satu tugas penting dalam pengujian perangkat lunak. Untuk merancang kasus uji dalam jumlah yang banyak dan menguji keseluruhan kasus uji tersebut, maka akan sangat dibutuhkan upaya yang cukup intensif dengan waktu yang lama. Pembentukan kasus uji yang dihasilkan secara otomatis akan dapat mengurangi biaya pengembangan perangkat lunak dan merupakan salah satu aspek yang paling kuat dalam pengujian secara otomatis (Singhal, 2012).

Kasus uji umumnya dirancang berdasarkan kode sumber. Kode sumber dapat dihasilkan setelah analisis dan perancangan perangkat lunak. Hal ini membuat uji coba kasus sulit terutama untuk pengujian pada tingkat cluster. Untuk menghindari pemborosan konsumsi waktu dan biaya yang digunakan dalam pengujian pada sistem berbasis kode, diharapkan dapat menghasilkan uji kasus pada tingkat desain sehingga keandalan perangkat lunak akan ditingkatkan ke tingkat yang optimal. Pengujian berbasis model lebih efisien dan efektif daripada pengujian berbasis kode. Untuk pembentukan kasus uji berdasarkan model, beberapa penelitian menggunakan diagram UML *(Unified Model Language)* yang berbeda-beda untuk menghasilkan kasus uji. Ada CASE Tools yang dapat mengkonversi diagram UML ke dalam file bertipe XMI (*XML Metadata Interchange*) ataupun XML (*Extensible Markup Language*). Akan tetapi tidak semua pengembang menggunakan UML dalam proses perancangan perangkat lunak. Banyak pengembang lain juga yang memanfaatkan model perancangan lain, seperti diagram alir data (*data flow diagram*), petri-net, ataupun diagram alir (*flowchart*).

Penelitian ini mencoba untuk menggunakan model lain yaitu diagram alir. Karakteristik pada diagram alir yang dapat dimanfaatkan untuk pembentukan kasus uji adalah diagram alir fokus terhadap penggambaran aliran proses dan aliran informasi. Diagram alir selalu diawali dan diakhiri oleh bagan terminator, sehingga akan mendukung pemodelan kasus uji ke dalam bentuk graf. Adanya terminator di awal dan akhir pada diagram alir cocok untuk pembentukan kasus uji. Dimana, semua kemungkinan kasus uji dicari dengan memperhatikan node awal dan akhir pada graf yang terbentuk. Dengan demikian, pembentukan kasus uji menggunakan diagram alir dapat terpenuhi. Diagram alir yang digunakan dalam penelitian ini, merupakan diagram alir yang mendeskripsikan setiap fungsi dari suatu kode sumber program. Berbeda dengan diagram UML, belum ada CASE Tools yang dapat mengkonversi diagram alir ke dalam file bertipe XMI (*XML Metadata Interchange*) ataupun XML (*Extensible Markup Language*). Diagram UML yang memiliki berbagai jenis diagram dan tentunya memiliki banyak notasi, diagram alir yang digunakan dalam penelitian hanya memiliki notasi untuk mendefinisikan proses, kondisi dan terminator. Dengan demikian, akan mempermudah dalam mengkonversi ke dalam metadata GML sebelum dilakukan konversi ke dalam bentuk graf. Variasi graf yang terbentuk hanya bergantung pada percabangan dan perulangan.

Penelitian ini berfokus pada pembentukan kasus uji berdasarkan diagram alir (*flowchart*) berdasarkan 3(tiga) kriteria cakupan yang sudah ditentukan, yaitu *Node Coverage, Transition Coverage, dan Simple Path Coverage*. Masukan yang digunakan dalam penelitian adalah diagram alir, dan keluaran yang dihasilkan adalah kasus uji. Setiap diagram alir menggambarkan sebuah fungsi dari kode sumber program. Selanjutnya, diagram alir dikonversi ke dalam bentuk *Flowchart Graph* (FG). Dari FG akan dihasilkan semua kemungkinan kasus uji (*test case*) beserta dengan nilai cakupan *Node Coverage, Transition Coverage, dan Simple Path Coverage.* Hasil dari ketiga kriteria cakupan tersebut dapat digunakan oleh tester untuk mengetahui kompleksitas tingkat pengujian.

Pengujian pada penelitian dirancang dengan tujuan membandingkan struktur kode program dengan kasus uji yang berhasil dibentuk. Hasil yang ingin dicapai adalah untuk mengetahui konsistensi antara *Program Execution Traces* (PET) saat kode program dijalankan dan kasus uji yang dibentuk berdasarkan diagram alir.

1.2. Perumusan Masalah

Rumusan masalah pada penelitian ini adalah sebagai berikut.

1. Bagaimana mengkonversi diagram alir ke dalam bentuk graf?
2. Bagaimana membentuk kasus uji sesuai dengan kriteria cakupan yang ditentukan yaitu *Node Coverage, Transition Coverage* dan *Simple Path Coverage*?
3. Bagaimana menganalisa hasil dari 3 kriteria cakupan (*Node Coverage, Transition Coverage* dan *Simple Path Coverage*) terhadap kompleksitas tingkat pengujian?

1.3. Batasan Masalah

Permasalahan yang dibahas pada penelitian ini memiliki beberapa batasan sebagai berikut.

1. Data uji yang digunakan dalam penelitian ini berupa kode sumber dengan paradigma pemrograman terstruktur menggunakan bahasa C#.
2. Kriteria cakupan pembentukan kasus uji pada penelitian ini diukur dengan menggunakan *Node Coverage*, *Transition Coverage* dan *Simple Path Coverage*.
3. Penetapan parameter-parameter inisialisasi awal adalah berdasarkan pada penelitian sebelumnya (Chen Minsong et al., 2007).

1.4. Tujuan

Tujuan yang akan dicapai dalam pembuatan tesis ini adalah menghasilkan kasus uji berdasarkan diagram alir sesuai dengan 3(tiga) kriteria cakupan yang sudah ditentukan. Pembentukan kasus uji berdasarkan diagram alir dapat diuji untuk mengetahui konsistensi antara *Program Execution Traces* (PET) saat kode program dijalankan dengan kasus uji yang dibentuk dari diagram alir. Selain mendeteksi adanya *bug* dan *error handling* dari fungsi yang dipanggil, ketidaksempurnaan dari *reverse engineering* ke dalam bentuk diagram alir juga dapat diketahui.

1.5. Manfaat Penelitian

Adapun manfaat dari penelitian ini adalah sebagai berikut.

1. Memperoleh gambaran tahapan dalam mengkonversi kode sumber program ke dalam bentuk diagram alir.
2. Memperoleh gambaran tahapan dalam memodelkan diagram alir ke dalam bentuk graf.
3. Membantu dalam mempermudah pengujian perangkat lunak sehingga dapat mengurangi waktu serta biaya dalam tahap pengujian.
4. Dapat membangun sebuah kakas bantu yang otomatis dalam membentuk kasus uji berdasarkan diagram alir.

1.6. Kontribusi Penelitian

Kontribusi penelitian ini adalah membentuk kasus uji berdasarkan diagram alir sesuai dengan 3(tiga) kriteria cakupan yang sudah ditentukan yaitu yaitu *Node Coverage*, *Transition Coverage* dan *Simple Path Coverage*. Masukan yang digunakan dalam penelitian adalah diagram alir, dan keluaran yang dihasilkan adalah kasus uji. Dalam penelitian ini kasus uji digambarkan berupa jalur uji yang dimodelkan ke dalam bentuk graf. Diagram alir tersebut dikonversi ke dalam bentuk graf untuk proses pelabelan terhadap instruksi di setiap baris kode program. Setiap node pada graf merepresentasikan sebuah instruksi pada kode sumber program.Sebuah graf dapat membentuk banyak kasus uji. Berdasarkan pengujian di awal, kasus uji yang dihasilkan tidak perlu dioptimasi. Hal ini dikarenakan satu kasus uji yang optimal tidak dapat merepresentasikan kasus uji yang lain, sehingga semua kasus uji harus uji. Oleh karena itu, penelitian berfokus hanya sampai pada tahap pembentukan kasus uji saja.

BAB 2
KAJIAN PUSTAKA DAN DASAR TEORI

Pada bab ini akan dijelaskan tentang pustaka yang terkait dengan landasan penelitian. Pustaka yang terkait adalah desain perangkat lunak, diagram alir, graf, pengujian perangkat lunak dan pembentukan kasus uji.

2.1. Desain Perangkat Lunak

Desain perangkat lunak adalah proses dimana kebutuhan diterjemahkan ke dalam representasi perangkat lunak. Desain perangkat lunakberada di bagianinti teknis rekayasa perangkat lunak dan diterapkan tanpa memperhatikanproses model perangkat lunak yang digunakan. Diawali dengan kebutuhan perangkat lunak yang telah dianalisis dan ditentukan, desain perangkat lunak adalah satu dari tiga kegiatan teknis yang diperlukan untuk membangun dan memverifikasi perangkat lunak. Tugas dari desain perangkat lunak adalah menghasilkan desain data, desain arsitektur, antarmuka desain, dan desain komponen.

Desain data mentransformasikan model domain informasi yang dibuat selama analisis ke dalam struktur data yang akan dibutuhkan untuk mengimplementasikan perangkat lunak. Desain arsitektur mendefinisikan hubungan antara elemen struktural utama perangkat lunak, pola desain yang dapat digunakan untuk mencapai kebutuhan yang sudah ditetapkan untuk sistem, dan kendala yang mempengaruhi cara dimana pola desain arsitektur dapat diterapkan. Desain antarmuka menjelaskan bagaimana perangkat lunak berkomunikasidengan dirinya sendiri, dengan sistem yang saling beroperasi, dan dengan manusia yang menggunakannya. Sebuah antarmuka menyiratkan alur informasi dan jenis tertentu perilaku. Desain tingkat komponen mentransformasikan elemen struktural dari arsitektur perangkat lunak menjadi deskripsi prosedural dari komponen perangkat lunak.

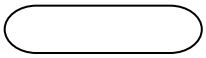
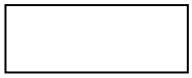
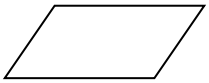
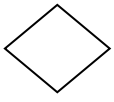
Teknik pemodelan data (Suryn, 2014) menyebutkan ada 5 pendekatan model, yaitu *floworiented models*, *behavioral models, class oriented models*, *scenariobased models*, dan *data models*. *Floworiented models* mendeskripsikan sistem dengan menitikberatkan pada alur data. Pada umumnya menggunakan diagram alir dan DFD (*Data Flow Diagram*). *Behavioral models* menggambarkan bagaimana perangkat lunak berperilaku sebagai konsekuensi dari peristiwa eksternal, menitikberatkan pada tingkah laku data (objek). Pada umumnya

menggunakan *Sequence* dan *State Diagram*. *Class oriented models* mendeskripsikan sistem dengan menitikberatkan pada klasifikasi data(objek). Pada umumnya menggunakan *Class Diagram*. *Scenario based models* mendeskripsikan sistem dengan menitikberatkan pada skenario sistem, umumnya digambarkan dengan *usecase, activity* dan *swimlane diagram*. Data models merupakan model data yang menggambarkan domain informasi untuk suatu permasalahan.

2.2. Diagram Alir

Diagram alir atau *Flowchart* adalah serangkaian bagan-bagan yang menggambarkan aliran program. Flowchart atau diagram alir memiliki bagan-bagan yang melambangkan fungsi tertentu (Umi Proboyekti, 2012). Bagan, nama dan fungsinya seperti yang disajikan pada Tabel 2.1.

Tabel 2.1 Notasi pada Diagram Alir

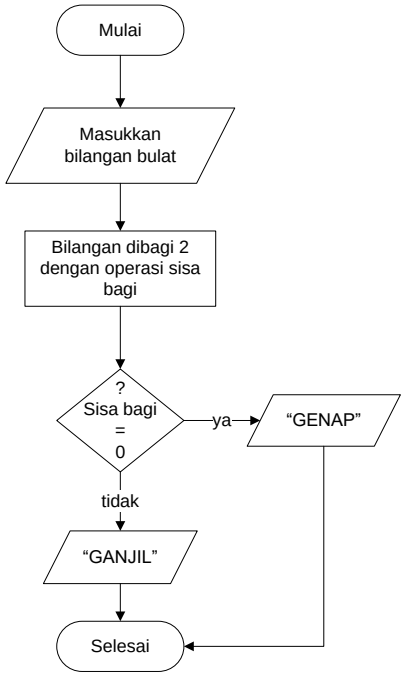
BAGAN	NAMA	FUNGSI
	**TERMINATOR**	Awal atau akhir program
	FLOW	Arah aliran program
	**DOKUMEN**	Input berasal dari dokumen dalam bentuk kertas
	**PROCESS**	Proses/pengolahan data
	**INPUT/OUTPUT DATA**	Input/output data
	**SUB PROGRAM**	Sub program
	**DECISION**	Seleksi/kondisi

Kemampuan untuk membangun, membaca, dan menafsirkan diagram alir adalah keterampilan yang diperlukan di semua teknologi instruksional. Simbol *flowchart* diakui secara universal di berbagai disiplin ilmu, dan akan memungkinkan desainer instruksional untuk mendapatkan informasi yang akurat,

efisien, dan ringkas.Selain itu, programmer, desainer perangkat lunak dan multimedia secara rutin menggunakan diagram alir untuk menggambarkan kemampuan perangkat lunak.

Flowchart selalu diawali dan diakhiri oleh bagan terminator. Aliran selalu dari ataske bawah, satu demi satu langkah. Tidak ada proses yang dikerjakan bersamaan, semua dikerjakan satu persatu.Proses yang dilakukan komputer sebenarnya hanya ada 3 proses: input, proses data dan output. Dengan demikian, ketika ada suatu masalah yang akan diselesaikan dengan suatu software, maka hal yang perlu diidentifikasi adalah input, proses data dan output.

Studi kasusnya adalah bagaimana menentukan bahwa suatu bilangan itu adalah bilangan genap atau ganjil. Masukannya merupakan bilangan bulat. Proses yang dilakukan adalah menentukan bilangan ganjil atau genap dengan melakukanpembagian bilangan dengan bilangan 2. Jika sisa pembagian NOL maka bilangan tersebut genap, sebaliknya ganjil. Keluarannya adalah bilangan ganjil atau bilangan genap.



Gambar 2.1 Contoh Studi Kasus Penentuan Bilangan Ganjil atau Genap

Ada beberapa jenis - jenis *flowchart* diantaranya:

1. Bagan alir sistem (*systems flowchart*).

   System flowchart dapat didefinisikan sebagai bagan yang menunjukkan arus pekerjaan secara keseluruhan dari sistem. Bagan ini menjelaskan urut-urutan dari prosedur-prosedur yang ada di dalam sistem. Bagan alir sistem menunjukkan apa yang dikerjakan di sistem.

2. Bagan alir dokumen (*document flowchart*).

   Bagan alir dokumen (*document flowchart*) atau disebut juga bagan alir formulir (*form flowchart*) atau paperwork flowchart merupakan bagan alir yang menunjukkan arus dari laporan dan formulir termasuk tembusan-tembusannya.

3. Bagan alir skematik (*schematic flowchart*).

   Bagan alir skematik (*schematic flowchart*) merupakan bagan alir yang mirip dengan bagan alir sistem, yaitu untuk menggambarkan prosedur di dalam sistem. Perbedaannya adalah, bagan alir skematik selain menggunakan simbol-simbol bagan alir sistem, juga menggunakan gambar-gambar komputer dan peralatan lainnya yang digunakan. Maksud penggunaan gambar-gambar ini adalah untuk memudahkan komunikasi kepada orang yang kurang paham dengan simbol-simbol bagan alir.

4. Bagan alir program (*program flowchart*).

   Bagan alir program (*program flowchart*) merupakan bagan yang menjelaskan secara rinci langkah-langkah dari proses program. Bagan alir program dibuat dari derivikasi bagan alir sistem. Bagan alir program dapat terdiri dari dua macam, yaitu bagan alir logika program (*program logic flowchart*) dan bagan alir program komputer terinci (*detailed computer program flowchart*). Bagan alir logika program digunakan untuk menggambarkan tiap-tiap langkah di dalam program komputer secara logika. Bagan alir program komputer terinci (*detailed computer program flow-chart*) digunakan untuk menggambarkan instruksi-instruksi program komputer secara terinci.

5. Bagan alir proses (*process flowchart*).

   Bagan alir proses (*process flowchart*) merupakan bagan alir yang banyak digunakan di teknik industri. Bagan alir ini juga berguna bagi analis sistem untuk menggambarkan proses dalam suatu prosedur.

2.3. Graf

Graf adalah pasangan himpunan (V, E), dan ditulis dengan notasi G = (V, E), V adalah himpunan tidak kosong dari verteks-verteks $\{v_1, v_2, \ldots, v_n\}$ yang dalam hal ini verteks merupakan himpunan tidak kosong dari verteks-verteks (*vertices* atau *node*) dan E adalah himpunan *edge* $\{e_1, e_2, \ldots, e_n\}$ atau sisi yang menghubungkan sepasang verteks.

(Munir, 2009) Sebuah graf dimungkinkan tidak mempunyai *edge* satu buah pun, tetapi verteksnya harus ada minimal satu. Graf yang hanya memiliki satu buah verteks tanpa sebuah *edge* pun dinamakan graf trivia.

Graf dapat dikelompokkan menjadi beberapa jenis sesuai dengan sudut pandang pengelompokannya. Pengelompokan graf dapat dipandang berdasarkan ada tidaknya rusuk ganda, berdasarkan jumlah simpul, atau berdasarkan orientasi arah pada rusuk (Munir, 2005:357).

Berdasarkan ada tidaknya gelang (loop) yaitu rusuk yang menghubungkan sebuah simpul dengan dirinya sendiri atau rusuk ganda pada suatu graf, maka secara umum graf dapat digolongkan menjadi dua jenis, graf sederhana dan graf tak sederhana.

Graf sederhana adalah graf yang tidak mempunyai rusuk ganda dan atau, gelang. Pada graf sederhana, rusuk adalah pasangan tak terurut (unordered pairs) (Harju, 2012). Jadi rusuk (u, v) sama dengan (v, u). Graf sederhana juga dapat didefinisikan sebagai G = (V, E), terdiri dari V, himpunan tidak kosong simpul-simpul dan E, himpunan pasangan tak terurut yang berbeda yang disebut rusuk. Berikut adalah contoh graf sederhana.

Graf yang mengandung rusuk ganda atau gelang dinamakan graf tak sederhana (*unsimple graph*). Ada dua macam graf tak sederhana, yaitu graf ganda (*multigraph*) atau graf semu (*pseudograph*). Graf ganda adalah graf yang mengandung rusuk ganda. Graf semu adalah graf yang mengandung gelang (*loop*).

Selain berdasarkan ada tidaknya rusuk ganda dan jumlah simpul pada suatu graf, graf juga dapat dikelompokkan berdasarkan orientasi arah pada rusuknya. Pengelompokan berdasarkan orientasi arah pada rusuknya digolongkan menjadi dua yaitu graf tak berarah dan graf berarah (Bondy, Murty, 1982).

Graf tak berarah adalah graf yang rusuknya tidak mempunyai orientasi arah. Urutan pasangan simpul yang dihubungkan oleh rusuk tidak diperhatikan. Jadi (V1, V2) = (V2, V1) adalah rusuk yang sama.

Graf berarah adalah graf yang setiap rusuknya memiliki orientasi arah. Rusuk pada graf berarah disebut busur (arc). Pada graf berarah, (u, v) dan(v,u)enyatakan dua buah busur yang berbeda. Jadi (u, v) ≠ (v, u). Untuk busur (u, v), simpul u dinamakan simpul asal (initial vertex) dan simpul v dinamakan simpul terminal (terminal vertek). Graf berarah ini seringkali dijadikan dasar dalam pembentukan model mengenai aliran proses, peta lalu lintas, sistem jaringan listrik, jaringan

telepon, analisis jejaring sosial, dan lain sebagainya. Pada graf berarah, adanya gelang diperbolehkan, tetapi rusuk ganda tidak.

2.4. Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan kegiatan utama dalam mengevaluasi perangkat lunak dengan tujuan untukmenemukan kesalahan. Proses dimana kebutuhan dan komponen sistem dilaksanakan dan dievaluasi secara manual atau automasi untuk mengetahui apakah sistem memenuhi kebutuhan yang ditentukan dan perbedaan antara hasil yang diharapkan dengan aktual yang sudah ditentukan (Hooda, Scholar, & Singh Chhillar, 2015).

Pengujian perangkat lunak adalah bagian paling penting dari siklus hidup pengembangan perangkat lunak. Pengujian perangkat lunak yang efisien akan memberikan kontribusi pada pengadaan yang konsisten dan berorientasi pada kualitas perangkat lunak yang baik, kepuasan pengguna, biaya pemeliharaan yang lebih rendah, hasil yang benar dan sistem yang dikembangkan dapat diandalkan. Perangkat lunak dapat diuji dengan berbagai teknik. Seperti juga diketahui bahwa pengujian perangkat lunak didukung oleh beragam kasus uji. Kasus uji digunakan untuk memanipulasi perangkat lunak dalam pemanfaatan sumber daya yang tepat dan meminimalkan biaya perangkat lunak serta mengurangi waktu pelaksanaan (Verma & Arora, 2016)..

Pengujian perangkat lunak menjadi tugas yang lebih dan lebih sulit pada saat ini, karena dalam siklus rekayasa perangkat lunak saat ini desain dan pengujian aktivitas dipisahkan, dimana hal ini dapat mengarah pada situasi dimana kasus uji tidak selaras dengan aplikasi yang sebenarnya. Salah satu cara untuk memecahkan masalah ini adalah untuk mengambil model dari aplikasi ke dalam penggunaan, misalnya, model antarmuka pengguna dan dari mana kasus uji dapat diturunkan secara otomatis. Teknik ini dikenal sebagai pengujian berbasis model.

Pengujian berbasis model merupakan teknik untuk pembentukan kasus uji dari model perangkat lunak tertentu. Keuntungan utama dari teknik ini adalah bahwa pembentukan kasus uji yang dilakukan secara sistematis akan mendapatkan semua kombinasi kasus uji yang terkait dengan kebutuhan, dimana kebutuhan tersebut diwakili dengan model untuk mengotomatisasi pengujian terhadap desain dan proses pelaksanaan pengujian. Dengan menghilangkan cacat yang ada pada model sebelum koding dimulai dan mengotomatisasi pembentukan kasus uji akan

dapat menghemat biaya secara signifikan dan menghasilkan kualitas kode yang lebih tinggi (Mikko Aleksi Mäkinen, 2007).

2.5. Pembentukan Kasus Uji

Pembentukan kasus uji adalah proses membangun rangkaian pengujian untuk mendeteksi kesalahan sistem dan merupakan proses yang paling penting serta mendasar dari pengujian perangkat lunak.Proses pengujian yang dilakukan secaraotomatis akan dapat mengurangi biaya pengembangan perangkat lunak. Hal ini tergantung pada kualitas atau nilai fitness dan jumlah kasus uji dijalankan.

Kasus uji merupakan masukan untuk program yang diuji. Kumpulan kondisi atau variabel dimana penguji akan menentukan apakah sebuah aplikasi atau sistem perangkat lunak bekerja dengan benar atau tidak. Dan juga merupakan mekanisme untuk menentukan apakah suatu program perangkat lunak atau telah lulus uji atau gagal.

Pembentukan data untuk pengujian perangkat lunak adalah proses identifikasi masukan program yang memenuhi kriteria pengujian. Ada dua pendekatan berbeda yang diguanakan, yaitu pendekatan berorientasi pada path dan pendekatan yang berorientasi pada tujuan. Biasanya, jumlah kasus uji yang diperlukan untuk mengembangkan perangkat lunakdan juga dapat menemukan kesalahan sebanyak mungkin. Jadi, optimasi kasus uji sangat diperlukan (Kumar, 2011).

Secara umum ada alasan untuk mengotomatisasi pembentukan kasus uji dalam pengujian perangkat lunak. Alasan yang paling penting adalah mengurangi biaya pengujian perangkat lunak dan mengurangi jumlah kasus uji.Efisiensi pembentukan kasus uji adalah langkah penting untuk menyederhanakan pekerjaan pengujian dan meningkatkan efisiensi pengujian. Pengujian berlangsung tidak efisien karena jumlah kasus uji cukup besar, sehingga beberapa algoritma otomasi diperlukan untuk mengoptimalkan kasus uji.

(Minsong Chen et al., 2007) melakukan penelitian terkait pembentukan kasus uji secara otomatis berdasarkan UML diagram aktivitas untuk program java. Kasus uji dibentuk berdasarkan 3(tiga) kriteria cakupan, yaitu Activity Coverage, Transition Coverage dan Simple Path Coverage. Selanjutnya program java tersebut dijalankan untuk mendapatkan Program Execution Traces (PET) untuk dicocokkan dengan perilaku diagram aktivitas. Tujuannya adalah untuk melakukan pengecekan terhadap konsistensi antara PET dengan perilaku diagram aktivitas.

(Jena et al., 2014) melakukan penelitian terkait pembentukan kasus uji menggunakan UML diagram aktivitas dan menerapkan algoritma genetika untuk menghasilkan kasus uji yang optimal.Metode yang diusulkan adalah diagram aktivitas dibentuk ke Activity Flow Table (AFT) dan kemudian dikonversi ke Activity Flow Graph (AFG). Kriteria cakupan sangat penting dalam pembentukan kasus uji. Dengan menggunakan kriteria cakupan, AFG dan kasus uji bisa dihasilkan. Algoritma genetika diterapkan untuk menghasilkan optimasi kasus uji.

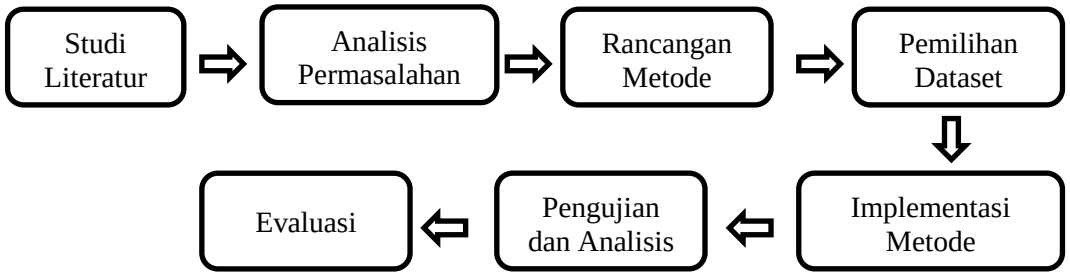
(Sumalatha, 2009) fokus pada pendekatan berbasi model untukpembentukan kasus uji secara otomatis.Pembentukan kasus uji untuk perangkat lunak berorientasi objek menggunakan diagram UML seperti diagram aktivitas. Penulis mencoba untuk mengoptimalkan jumlah kasus uji menggunakan algoritma evolusi seperti algoritma genetika.

(Sumalatha, 2013) mengusulkan untuk pembentukan kasus uji pada perangkat lunak berorientasi objek menggunakan diagram UML seperti sequence diagram. Kasus uji dioptimalkan menggunakan algoritma evolusi yaitu algoritma genetika. Metode ini merupakan pembentukankasus uji yangmungkin dapat menginspirasi para pengembang sistem untuk meningkatkan kualitas desain dan menemukan beberapa kasus uji untuk siap dieksekusi.

(Shanthi & Kumar, 2012) mengusulkan pendekatan baru untuk pengujian perangkat lunak pada tahap awal, sehingga akan mudah bagi penguji perangkat lunak untuk menguji perangkat lunak pada tahap selanjutnya. Penelitian ini berfokus padapembentukan kasus ujiberdasarkan UML sequence diagram menggunakan algoritma genetika, dimana kasus uji terbaik dioptimasi dan kasus uji divalidasi dengan prioritas.

BAB 3
METODOLOGI PENELITIAN

Bab ini akan memaparkan tentang metodologi penelitian yang digunakan pada penelitian ini, yang terdiri dari (1) Studi Literatur, (2) Analisis Permasalahan, dan (3) Rancangan Metode (4) Pemilihan Dataset (5) Implementasi Metode (6) Pengujia dan Analisis Hasil (7) Evaluasi. Ilustrasi alur metodologi penelitian dapat dilihat pada Gambar 3.1.



Gambar 3.1 Alur Metodologi Penelitian

Penjelasan tahapan metode penelitian pada Gambar 3.1 akan diterangkan secara terperinci pada subbab berikut.

3.1. Studi Literatur

Penelitian diawali dengan melakukan kajian yang berkaitan dengan topik penelitian. Referensi yang digunakan dalam penelitian ini berasal dari jurnal, konferensi, dan buku yang berkaitan dengan optimasi kasus uji berdasarkan diagram alir menggunakan algoritma genetika. Berdasarkan studi literatur yang telah dilakukan, terdapat informasi sebagai berikut.

Untuk dapat memvalidasi kebutuhan pengguna pada perangkat lunak, salah satu cara yang dapat dilakukan adalah dengan melakukan pengujian terhadap perangkat lunak yang sedang dikembangkan. Pengujian dapat dilakukan dengan melakukan pembentukan kasus uji.

Pembentukan kasus uji secara otomatis dapat membantu pengembang perangkat lunak dalam mengurangi biaya dan waktu pengujian. Banyak penelitian terkait pembentukan kasus uji pada pengujian berdasarkan model, peneliti memilih untuk menggunakan model UML.

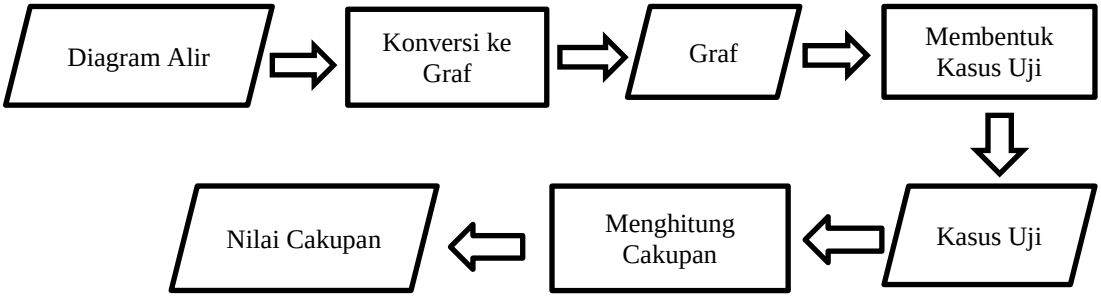
3.2. Analisis Permasalahan

Tujuan utama dari pengujian adalah meminimalkan jumlah kasus uji untuk menghemat waktu dan sumber daya. Waktu yang dibutuhkan untuk pengujian bergantung pada besar dan kompleksnya perangkat lunak yang dikembangkan.

Diharapkan tester dapat menghasilkan uji kasus pada tingkat desain sehingga keandalan perangkat lunak akan ditingkatkan ke tingkat yang optimal. Pengujian berbasis model lebih efisien dan efektif daripada pengujian berbasis kode. Untuk pembentukan kasus uji berdasarkan model, beberapa penelitian menggunakan diagram UML *(Unified Model Language)* yang berbeda-beda untuk menghasilkan kasus uji. Akan tetapi tidak semua pengembang menggunakan UML dalam proses perancangan perangkat lunak. Banyak pengembang lain juga yang memanfaatkan model perancangan lain, seperti diagram alur data (*data flow diagram*), petri-net, ataupun diagram alir (*flowchart*). Dalam penelitian ini kasus uji dibentuk berdasarkan diagram alir sesuai dengan tiga kriteria cakupan, yaitu *Node Coverage*, *Transition Coverage* dan *Simple Path Coverage*.

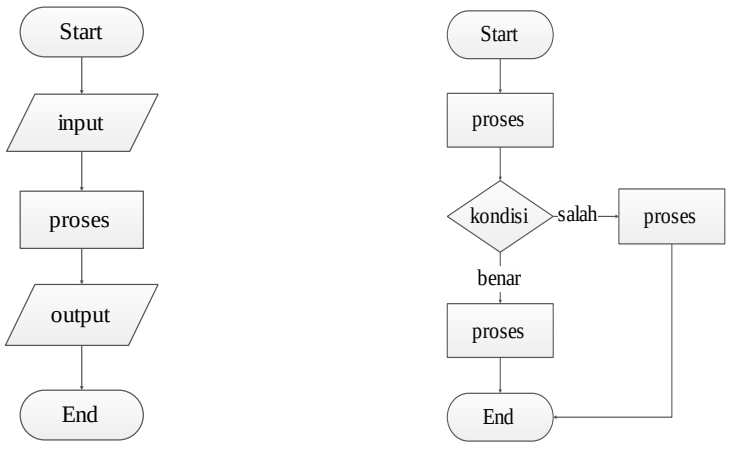
3.3. Rancangan Metode

Alur sistem pembentukan kasus uji dapat dilihat pada Gambar 3.2 dimana pembentukan kasus uji berdasarkan diagram alir dibagi menjadi empat tahap, yaitu (1) Diagram alir sebagai masukan (2) Konversi diagram alir ke dalam bentuk graf (3) Pembentukan kasus uji (4) Menghitung kriteria cakupan.

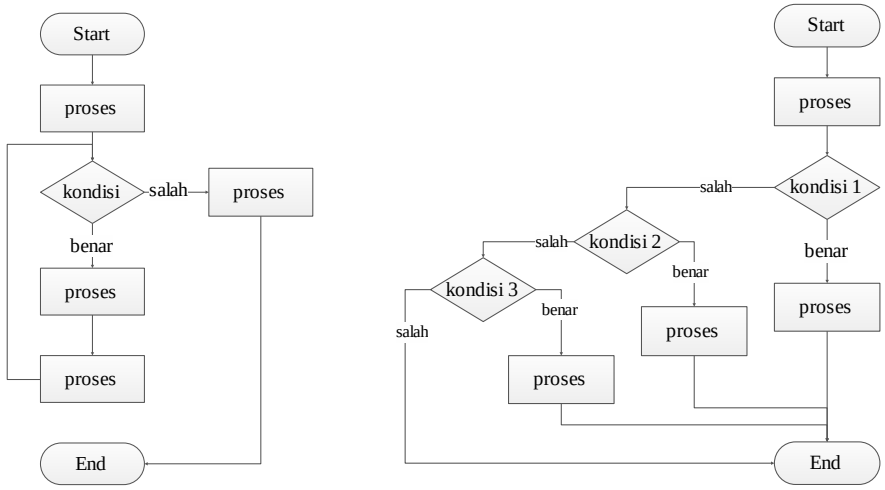


Gambar 3.2 Langkah Pembentukan Kasus Uji

Dalam penelitian ini, masukan yang digunakan adalah diagram alir. Mengacu pada Tabel 2.1, berikut diberikan contoh bentuk diagram alir secara umum pada Gambar 3.3.



(a) (b)

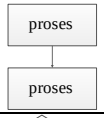
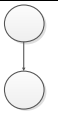
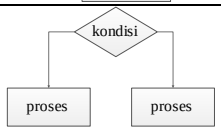
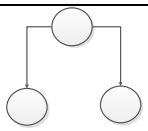


(c) (d)

Gambar 3.3 Bentuk Umum Diagram Alir

Proses selanjutnya adalah konversi diagram alir ke dalam bentuk graf. Berikut diberikan gambaran umum terkait aturan konversi dari diagram alir ke dalam bentuk graf pada Tabel 3.1.

Tabel 3.1 Tabel Konversi Simbol Diagram Alir ke Simbol Graf

Simbol Diagram Alir	Simbol Graf
	
	
	

Setelah graf terbentuk, maka dari graf tersebut akan dicari semua kemungkinan kasus uji menggunakan strategi DFS (*Depth First Search*) untuk mencegah adanya kasus uji yang redundan dan memastikan semua node pada graf sudah dikunjungi. Pencarian kasus uji ini dilakukan dengan memperhatikan node awal dan node akhir pada graf. Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada node sebelah kanan.

Setelah kasus uji berhasil dibentuk, maka selanjutnya adalah menghitung nilai ketiga kriteria cakupan yang sudah ditentukan, yaitu *Node Coverage, Transition Coverage* dan *Simple Path Coverage*. Tujuannya adalah untuk mengetahui kompleksitas tingkat pengujian.

3.4. Pemilihan Dataset

Diagram alir dibangun dengan melakukan proses rekayasa balik dari kode sumber yang dipilih sebagai dataset. Kakas bantu yang digunakan dalam penelitian ini antara lain adalah sebagai berikut.

1. Notepad++ yaitu kakas bantu yang digunakan untuk membangun metadata berdasarkan diagram alir ke dalam sebuah file bertipe GML.
2. Microsoft Visual Studio 2015 yang digunakan untuk mengkonversi metadata diagram alir ke dalam bentuk graf dan menghitung nilai cakupan yang telah ditentukan.

3.5. Implementasi Metode

Algoritma yang diusulkan untuk pembentukan kasus uji pada penelitian ini adalah sebagai berikut.

1. Diagram alir/*flowchart* sebagai masukan.
2. Mengkonversi diagram alir ke dalam bentuk file metadata bertipe GML.
3. Mengkonversi GML ke dalam bentuk *FlowchartGraph* (FG).
4. Menghasilkan semua kemungkinan kasus uji.
5. Menghitung kriteria cakupan yang telah ditentukan, yaitu *Node Coverage, Transition Coverage* dan *Simple Path Coverage*.

Penjelasan tentang kegiatan Implementasi Metode akan dibahas pada subbab berikut.

3.5.1 Diagram alir/*flowchart* sebagai masukan

Pada penelitian ini, masukan yang digunakan adalah diagram alir. Studi kasus yang digunakan adalah bentuk umum dari diagram alir yang mengandung satu percabangan seperti pada Gambar 3.3 (b).

3.5.2 Mengkonversi diagram alir ke dalam bentuk file metadata bertipe GML.

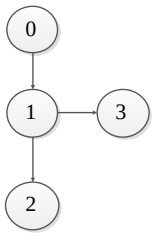
Untuk dapat dikonversi ke dalam bentuk graf, maka diagram alir yang digunakan sebagai masukan dikonversi secara manual ke dalam file metadata yang bertipe GML (*Geography Markup Language*). Langkah konversi ini dilakukan menggunakan kakas bantu notepad++. Untuk format penulisan file GML, yang perlu diperhatikan adalah *node, edge* dan *value*. *Node* mendefinisikan setiap instruksi yang terdapat pada flowchart. *Edge* mendefinisikan hubungan antara node. *Value* digunakan untuk mendefinisikan nilai dari node. File metadata tersebut dapat dilihat pada Gambar 3.4.

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed" parse.nodes="7" parse.edges="7"
        parse.order="nodesfirst" parse.nodeids="free" parse.edgeids="free">
        <nodes>
                <node id="0" value="proses 1"/>
                <node id="1" value="kondisi"/>
                <node id="2" value="proses 2"/>
                <node id="3" value="proses 3"/>
        </nodes>
        <edges>
                <edge id="0to1" source="0" target="1" />
                <edge id="1to2" source="1" target="2" />
                <edge id="1to3" source="1" target="3" />
        </edges>
  </graph>
</graphml>
```

Gambar 3.**4** File Metadata Bertipe GML

3.5.3 Mengkonversi GML ke dalam bentuk *FlowchartGraph*

Langkah berikutnya adalah mengkonversi file GML ke dalam bentuk *Flow Graph* yang dilakukan menggunakan kakas bantu Visual Studio 2015 seperti pada Gambar 3.5. Tujuannya adalah untuk proses pelabelan (*value* pada *node*). Setiap node tersebut mewakili setiap pernyataan pada baris kode sumber. Graf yang dihasilkan tidak digambarkan secara visual di dalam aplikasi yang digunakan. Akan tetapi, graf yang dihasilkan tersebut didefinisikan dengan menampilkan jumlah *node* beserta *value*, dan jumlah *edges* beserta relasi antar *node.*



Gambar 3.5 *Flow Graph* Contoh Studi Kasus

3.5.4 Membentuk kasus uji

Pada tahap ini, graf yang berhasil dibentuk akan diproses untuk menghasilkan semua kemungkinan kasus uji. Pembentukan kasus uji dilakukan menggunakan strategi DFS (*Depth First Search*) untuk mencegah adanya kasus uji yang redundan dan memastikan semua node pada graf sudah dikunjungi. Kasus uji yang dihasilkan mencakup semua cabang, kondisi dan perulangan. Berikut adalah semua kemungkinan kasus uji yang dihasilkan berdasadrkan contoh pada Gambar 3.5. Kasus uji 1 adalah 0->1->2 dan kasus uji 2 adalah 0->1->3.

3.5.5 Menghitung kriteria cakupan

Kriteria cakupan yang digunakan dalam penelitian ini ada 3 (tiga) yaitu *Node Coverage, Transition Coverage* dan *Simple Path Coverage*. *Node Coverage* adalah perhitungan nilai cakupan berdasarkan jumlah node yang terdapat pada kasus uji. *Transition Coverage* adalah perhitungan nilai cakupan berdasarkan jumlah transisi yang terdapat pada kasus uji. *Simple Path Coverage* adalah perhitungan nilai cakupan berdasarkan jumlah alur uji yang tidak mengandung perulangan yang terdapat pada graf.

a) Node Coverage

Perhitungan nilai cakupan berdasarkan jumlah node yang terdapat pada kasus uji. Nilai cakupan dari *Node Coverage* merupakan rasio node yang berhasil tercakup dengan keseluruhan node pada graf sesuai dengan Persamaan 3.1.

$$\frac{jumlah\ node\ yang\ tercakup}{jumlah\ keseluruhan\ node}\ x\ 100\% \qquad (3.1)$$

b) Transition Coverage

Perhitungan nilai cakupan berdasarkan jumlah transisi yang terdapat pada kasus uji. Nilai cakupan dari *Transition Coverage* merupakan rasio transisi yang berhasil tercakup dengan keseluruhan transisi pada graf sesuai dengan Persamaan 3.2.

$$\frac{jumlah\ transisi\ yang\ tercakup}{jumlah\ keseluruhan\ transisi}\ x\ 100\% \qquad (3.2)$$

c) Simple Path Coverage

Perhitungan nilai cakupan berdasarkan jumlah alur uji yang tidak mengandung perulangan yang terdapat pada graf. Nilai cakupan dari S*imple Path Coverage* merupakan rasio *simple path* yang berhasil tercakup dengan keseluruhan *simple path* pada graf sesuai dengan Persamaan 3.3.

$$\frac{jumlah\ simple\ path\ yang\ tercakup}{jumlah\ keseluruhan\ simple\ path}\ x\ 100\% \qquad (3.3)$$

Pada kakasi bantu yang digunakan, akan ditampilkan jumlah node, transisi, kasus uji, nilai dari *Node Coverage*, nilai dari *Transition Coverage*, dan *Simple Path Coverage*. Berdasarkan contoh pada Gambar 3.5, berhasil terbentuk 2 kasus uji. Hasil DFS dari kasus uji tersebut adalah 0->1->2 dan 0->1->3. Sehingga dapat diketahui bahwa alur kasus uji pertama sebagai berikut.

node id="0" value="proses 1"/>

<node id="1" value="kondisi"/>

<node id="2" value="proses 2"/>

Sedangkan alur kasus uji kedua sebagai berikut.

<node id="0" value="proses 1"/>

<node id="1" value="kondisi"/>

<node id="3" value="proses 3"/>

Selanjutnya dihitung ketiga kriteria cakupan yang sudah ditentukan, yaitu :

a) Node Coverage

Node yang tercakup adalah 3. Jumlah keseluruhan node adalah 4. Nilai cakupan = $\frac{3}{4}$x 100% = 75%.

b) Transition Coverage

Transisi yang tercakup adalah 2. Jumlah keseluruhan transisi adalah 3. Nilai cakupan = $\frac{2}{3}$x 100% = 67%.

c) Simple Path Coverage

Simple path yang terbentuk adalah 2, yaitu 0->1->2 dan 0->1->3. Nilai cakupan = $\frac{2}{2}$ x 100% = 100%.

3.6. Analisis Pengujian

Penelitian ini membandingkan teknik pengujian kotak putih pada struktur kode program dengan kasus uji yang dibentuk berdasarkan diagram alir. Hasil yang ingin dicapai adalah untuk memperoleh konsistensi antara *Program Execution Traces* (PET) saat kode program dijalankan dan kasus uji yang dibentuk dari diagram alir. Selain mendeteksi adanya *bug* dan *error handling* dari fungsi yang yang dipanggil, ketidaksempurnaan dari *reverse engineering* ke dalam bentuk diagram alir juga dapat diketahui.

[Halaman ini sengaja dikosongkan]

BAB 4
HASIL DAN PEMBAHASAN

Pada bab ini akan dibagi menjadi tiga bagian yaitu implementasi penelitian, uji coba, dan analisis uji coba. Lingkungan uji coba meliputi hardware dan software yang digunakan untuk implementasi system.

4.1. Implementasi Penelitian

Penelitian ini akan dibangun dalam lingkungan pengembangan sebagai berikut:

Sistem operasi : Windows 1064 bit

RAM : 4 GB

Processor : Intel Core i5

IDE : Microsoft Visual Studio 2008

4.2. Perancangan Uji Coba

Pada sub bab ini menjelaskan tentang pembagian dataset, skenario uji coba, sampel proses implementasi teknik pengujian serta analisa dan evaluasi dari hasil pengujian yang dilakukan.

4.2.1. Pembagian Dataset

Dataset yang digunakan pada penelitian ini adalah 10 kode sumber program console sederhana yang dikembengkan dengan bahasa pemrograman C# dan dapat dijalankan menggunakan di Microsoft Visual Studio 2008.

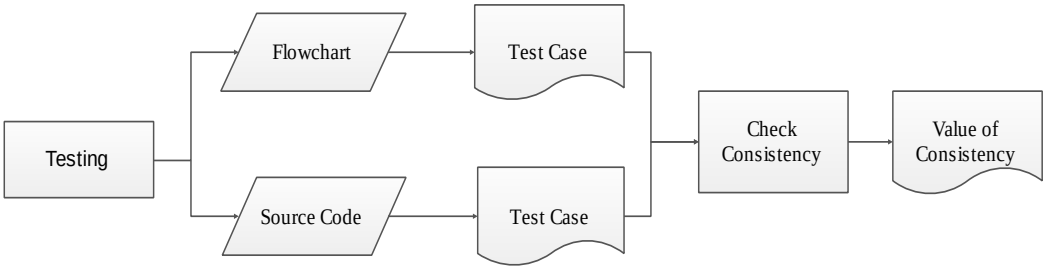
Tabel 4.1 Kode sumber program yang digunakan

No	Program Console
1	Menghitung penjumlahan, pengurangan, perkalian, dan pembagian dari 2 buah variabel yang diinputkan
2	Menentukan bilangan genap dan ganjil sebuah bilangan
3	Menentukan predikat suatu nilai
4	Menentukan nilai terbesar dari 3 buah bilangan
5	Menghitung luas persegi dan lingkaran
6	Program ATM
7	Perulangan menampilkan bilangan dari 1 sampai 10
8	Menampilkan bilangan genap dari 1 sampai 10
9	Menghitung jumlah bilangan yang diinputkan
10	Menghitung nilai rata- rata

Masukan yang digunakan dalam penelitian adalah diagram alir, dan keluaran yang dihasilkan adalah kasus uji. Diagram alir tersebut dirancang dari kode sumber program *console* sederhana menggunakan bahasa pemrograman C#. Selanjutnya, diagram alir tersebut dikonversi ke dalam bentuk graf untuk proses pelabelan terhadap instruksi di setiap baris kode program. Dalam penelitian ini kasus uji digambarkan berupa jalur uji yang dimodelkan ke dalam bentuk graf. Setiap node pada graf merepresentasikan sebuah instruksi pada kode sumber program.Sebuah graf dapat membentuk banyak kasus uji. Tahapan selanjutnya adalah melakukan pengujian terhadap konsistensi antara *Program Execution Traces* (PET) dengan kasus uji yang dibentuk secara otomatis. Pengujian dilakukan secara manual, yaitu dengan menjalankan kode sumber program dan memasukkan data uji yang bernilai valid dan tidak valid. Dengan demikian dapat diketahui bahwa kasus uji yang dibentuk secara otomatis konsisten terhadap kode sumber program. Selain itu juga dapat digunakan untuk mendeteksi adanya *bug* dan *error handling* dari fungsi yang dipanggil, serta mengetahui ketidaksempurnaan dari *reverse engineering* ke dalam bentuk diagram alir.

4.2.2. Skenario Uji Coba

Berdasarkan metode yang diusulkan, penelitian ini memiliki beberapa tahapan proses. Gambaran skenario pengujian untuk penelitian ini dapat dideskripsikan pada Gambar 4.1.



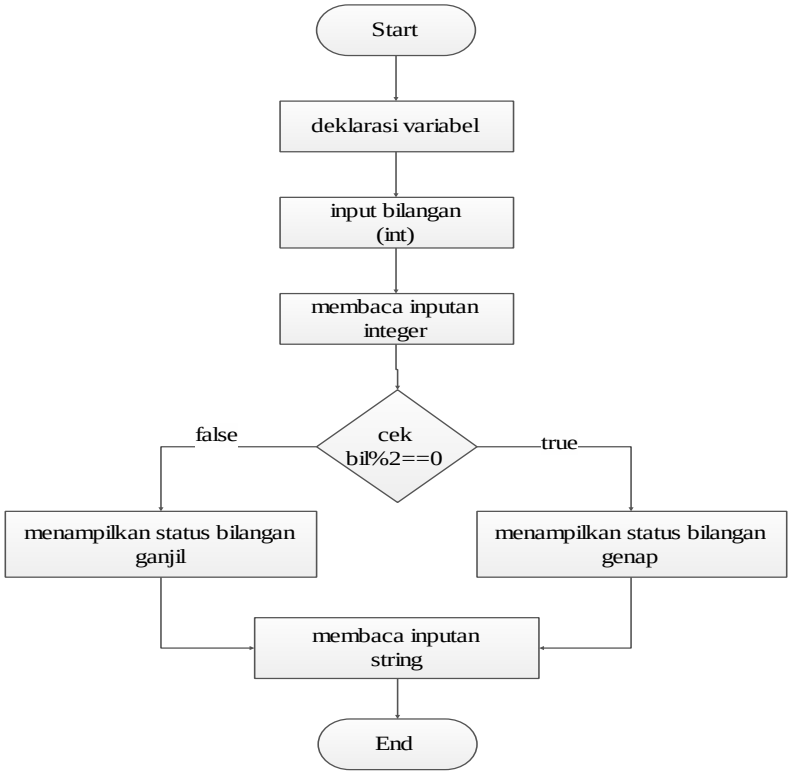
Gambar 4.1 Skenario Uji Coba

4.2.3. Implementasi Teknik Pengujian

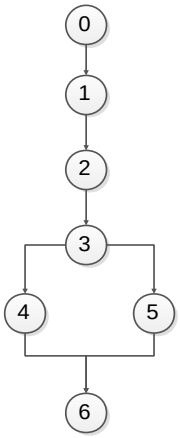
Sub bab ini akan menjelaskan salah satu contoh implementasi teknik pengujian yang dilakukan seperti yang ditunjukkan pada Gambar 4.1. Dataset yang digunakan dalam penelitian ini berupa kode sumber dari sebuah program *console* sederhana seperti pada contoh Gambar 4.2.. Sub bab 4.2.3.1 menjelaskan pembentukan kasus uji yang digunakan dalam penelitian ini.

4.2.3.1. Pembentukan Kasus Uji 1

Gambar 4.2 merupakan diagram alir yang digunakan sebagai contoh untuk merepresentasikan bentuk umum dari diagram alir yang memiliki satu percabangan. Kasus uji dibentuk dari hasil konversi diagram alir ke dalam bentuk graf. Untuk dapat dikonversi ke dalam bentuk graf, maka diagram alir dimodelkan ke dalam bentuk metadata yang bertipe GML (*Geography Markup Language*). Langkah konversi ini dilakukan menggunakan kakas bantu notepad++. Pada tahap ini, graf yang berhasil dibentuk akan diproses untuk menghasilkan semua kemungkinan kasus uji beserta dengan ketiga nilai cakupannya, yaitu *Node Coverage*, *Transition Coverage* dan *Simple Path Coverage*.



Gambar 4.2 Diagram Alir Menentukan Bilangan Ganjil atau Genap



Gambar 4.3 Graf Hasil Konversi Diagram Alir

4.2.3.2. Menghitung Kriteria Cakupan dari Kasus Uji 1

Dalam penelitian ini sudah ditentukan 3 (tiga) kriteria cakupan yaitu *Node Coverage*, *Transition Coverage* dan *Simple Path Coverage*. Berikut adalah penjelasan dan langkah-langkah yang dilakukan untuk menghitung kasus uji yang sesuai dengan kriteria cakupan.

Node Coverage adalah perhitungan nilai cakupan berdasarkan jumlah node yang terdapat pada kasus uji. *Transition Coverage* adalah perhitungan nilai cakupan berdasarkan jumlah transisi yang terdapat pada kasus uji. *Simple Path Coverage* adalah perhitungan nilai cakupan berdasarkan jumlah alur uji yang tidak mengandung perulangan yang terdapat pada graf.

Graf yang sudah terbentuk dapat dilihat pada Gambar 4.3. Langkah selanjutnya adalah menghitung berapa jumlah node, jumlah transisi dan jumlah *simple path* pada graf tersebut. Jumlah node adalah 7, jumlah transisi adalah 7 dan jumlah *simple path* adalah 2. Kasus uji yang terbentuk adalah 0->1->2->3->5->6 dan 0->1->2->3->4->6. Selanjutnya, dihitung nilai cakupannya sesuai dengan kriteria yang sudah ditentukan.

1. *Node Coverage*

   Nilai cakupan dari *Node Coverage* merupakan rasio node yang berhasil tercakup dengan keseluruhan jumlah node pada graf dikalikan 100%.

   0->1->2->4->5->6. Node yang tercakup adalah 6. Jumlah keseluruhan node adalah 7. Nilai cakupan = $\frac{6}{7}$ x 100% = 85%

2. *Transition Coverage*

   Nilai cakupan dari *Transition Coverage* merupakan rasio transisi yang berhasil tercakup dengan keseluruhan jumlah transisi pada graf.

   0->1->2->4->5->6. Transisi yang tercakup adalah 5. Jumlah keseluruhan transisi adalah7. Nilai cakupan = $\frac{5}{7}$ x 100% = 71%

3. *Simple Path Coverage*

   Nilai cakupan dari *Simple Path* yang tidak mengandung perulangan dan merupakan rasio simple path yang berhasil tercakup dengan keseluruhan *simple path* pada graf. Dalam contoh kasus yang diberikan, *simple path* yang terbentuk adalah 2. Dan semua kemungkinan kasus uji yang diberikan pada contoh studi kasus adalah 2 *simple path*. Nilai cakupan = $\frac{2}{2}$ x 100% = 100%

4.2.3.3. Pengujian Konsistensi Antara PET dengan Kasus Uji 1

Kasus uji yang terbentuk berdasarkan Gambar 4.3 sejumlah 2. Urutan langkah kasus uji yang pertama adalah 0->1->2->3->4->6 adalah sebagai berikut.

<node id="0" value="deklarasi variabel"/>

<node id="1" value="input bilangan (int)"/>

<node id="2" value="membaca inputan integer"/>

<node id="3" value="cek bil % 2 == 0"/>

<node id="4" value="menampilkan status bilangan ganjl);"/>

<node id="6" value="membaca inputan string"/>

Urutan langkah kasus uji yang pertama adalah 0->1->2->3->5->6 adalah sebagai berikut.

<node id="0" value="deklarasi variabel"/>

<node id="1" value="input bilangan (int)"/>

<node id="2" value="membaca inputan integer"/>

<node id="3" value="cek bil % 2 == 0"/>

<node id="5" value="menampilkan status bilangan genap);"/>

<node id="6" value="membaca inputan string"/>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int bil;
            System.Console.Write("Input Bilangan = ");
            bil = int.Parse(System.Console.ReadLine());

            if (bil % 2 == 0)
            System.Console.WriteLine("Bil  "  +  bil  +  "  adalah
            genap");
            else
            System.Console.WriteLine("Bil  "  +  bil  +  "  adalah
            ganjil");
            System.Console.ReadLine();
        }
    }
}
```

Gambar 4.4 Kode Sumber Program Console Menentukan Bilangan Ganjil atau Genap

Pengujian pada penelitian dirancang dengan tujuan membandingkan struktur kode program dengan kasus uji yang berhasil dibentuk. Hasil yang ingin dicapai adalah untuk mengetahui konsistensi antara *Program Execution Traces* (PET) saat kode program dijalankan dengan kasus uji yang berhasil dibentuk.

Program Execution Traces (PET) dilakukan terhadap kode sumber pada Gambar 4.4 untuk kasus uji yang menunjukkan bilangan ganjil dengan data uji yang sudah ditentukan, hasilnya adalah sebagai berikut.

Step 1 -> int bil;
Step 2 -> System.Console.Write("Input Bilangan = ");
Step 3 -> bil = int.Parse(System.Console.ReadLine());
Step 4 -> if (bil % 2 == 0)
Step 5 -> System.Console.WriteLine("Bil " + bil + " adalah ganjil");
Step 6 -> System.Console.ReadLine();

Program Execution Traces (PET) dilakukan terhadap kode sumber pada Gambar 4.4 untuk kasus uji yang menunjukkan bilangan genap dengan data uji yang sudah ditentukan, hasilnya adalah sebagai berikut.

Step 1 -> int bil;
Step 2 -> System.Console.Write("Input Bilangan = ");
Step 3 -> bil = int.Parse(System.Console.ReadLine());
Step 4 -> if (bil % 2 == 0)
Step 5 -> System.Console.WriteLine ("Bil " + bil + " adalah genap");
Step 6 -> System.Console.ReadLine();

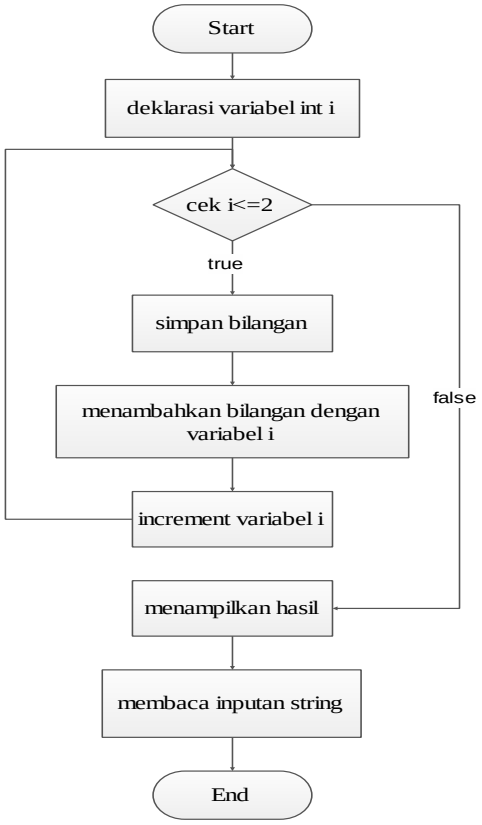
Dari hasil pengujian contoh studi kasus didapatkan hasil seperti yang sudah dijabarkan pada Tabel 4.2. Dilihat dari tahapan pengujian yang dilakukan bahwa ketika data uji dimasukkan untuk bilangan ganjil ataupun genap, antara alur dari kasus uji dengan *Program Execution Traces* (PET) adalah sesuai. Hal ini dapat diartikan bahwa kasus uji yang dihasilkan konsisten terhadap *Program Execution Traces* (PET).

Tabel 4.2 Hasil Pengujian Terhadap Diagram Alir 1

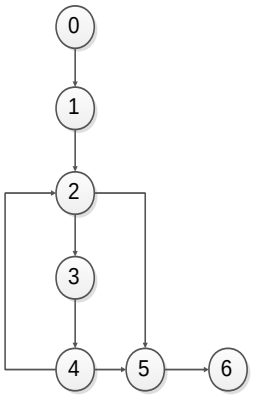
No		Masukan	Keluaran	Kasus Uji
1	VALID	0	Bilangan genap	Konsisten
2		-5	Bilangan ganjil	Konsisten
3		-100	Bilangan genap	Konsisten
4		2147483647	Bilangan ganjil	Konsisten
5		-2147483648	Bilangan genap	Konsisten

4.2.3.4. Pembentukan Kasus Uji 2

Gambar 4.6 merupakan diagram alir dari kode sumber program merepresentasikan bentuk umum dari diagram alir yang mengandung perulangan for. Kasus uji dibentuk dari hasil konversi diagram alir ke dalam bentuk graf. Untuk dapat dikonversi ke dalam bentuk graf, maka diagram alir dimodelkan ke dalam bentuk metadata yang bertipe GML (*Geography Markup Language*). Langkah konversi ini dilakukan menggunakan kakas bantu notepad++. Pada tahap ini, graf yang berhasil dibentuk akan diproses untuk menghasilkan semua kemungkinan kasus uji beserta dengan ketiga nilai cakupannya, yaitu *Node Coverage*, *Transition Coverage* dan *Simple Path Coverage*.



Gambar 4.5 Diagram Alir Perulangan Menampilkan Bilangan 1 Sampai 2



Gambar 4.6 Flow Graph Perulangan Menampilkan Bilangan 1 Sampai 2

4.2.3.5. Menghitung Kriteria Cakupan dari Kasus Uji 2

Dalam penelitian ini sudah ditentukan 3 (tiga) kriteria cakupan yaitu *Node Coverage*, *Transition Coverage* dan *Simple Path Coverage*. Berikut adalah penjelasan dan langkah-langkah yang dilakukan untuk menghitung kasus uji yang sesuai dengan kriteria cakupan.

Node Coverage adalah perhitungan nilai cakupan berdasarkan jumlah node yang terdapat pada kasus uji. *Transition Coverage* adalah perhitungan nilai cakupan berdasarkan jumlah transisi yang terdapat pada kasus uji. *Simple Path Coverage* adalah perhitungan nilai cakupan berdasarkan jumlah alur uji yang tidak mengandung perulangan yang terdapat pada graf.

Graf yang sudah terbentuk dapat dilihat pada Gambar 4.6. Langkah selanjutnya adalah menghitung berapa jumlah node, jumlah transisi dan jumlah *simple path* pada graf tersebut. Jumlah node adalah 7, jumlah transisi adalah 8 dan jumlah *simple path* adalah 1. Kasus uji yang terbentuk adalah 0->1->2->3->4->5 ->6 dan 0->1->2->6->7. Selanjutnya, dihitung nilai cakupannya sesuai dengan kriteria yang sudah ditentukan.

1. *Node Coverage*

   Nilai cakupan dari *Node Coverage* merupakan rasio node yang berhasil tercakup dengan keseluruhan jumlah node pada graf dikalikan 100%.

   0->1->2->6->7. Node yang tercakup adalah 6. Jumlah keseluruhan node adalah 7. Nilai cakupan = $\frac{6}{7}$ x 100% = 85%

2. *Transition Coverage*

   Nilai cakupan dari *Transition Coverage* merupakan rasio transisi yang berhasil tercakup dengan keseluruhan jumlah transisi pada graf.

   0->1->2->6->7. Transisi yang tercakup adalah 5. Jumlah keseluruhan transisi adalah 8. Nilai cakupan = $\frac{5}{8}$ x 100% = 62%

3. *Simple Path Coverage*

   Nilai cakupan dari *Simple Path* yang tidak mengandung perulangan dan merupakan rasio simple path yang berhasil tercakup dengan keseluruhan *simple path* pada graf. Dalam contoh kasus yang diberikan, simple path yang terbentuk adalah 1 yaitu 0->1->2->6->7. Dan semua kemungkinan kasus uji yang diberikan pada contoh studi kasus adalah 2 *simple path*.

   Nilai cakupan = $\frac{1}{2}$ x 100% = 50%.

4.2.3.6. Pengujian Konsistensi Antara PET dengan Kasus Uji 2

Kasus uji yang terbentuk berdasarkan Gambar 4.6 sejumlah 2. Urutan langkah kasus uji yang pertama adalah 0->1->2->3->4->5->6->7sebagai berikut.

<node id="0" value="deklarasi variabel integer" />

<node id="1" value="menampilkan tulisan Bilangan =" />

<node id="2" value="cek i<=2 "/>

<node id="3" value="Simpan bilangan"/>

<node id="4" value="menambahkan bilangan dengan variabel i"/>

<node id="5" value="increment variabel i"/>

<node id="6" value="menampilkan hasil" />

<node id="7" value="membaca inputan string" />

Urutan langkah kasus uji yang pertama adalah 0->1->2->6->7 adalah sebagai berikut

<node id="0" value="deklarasi variabel integer " />

<node id="1" value="menampilkan tulisan Bilangan =" />

<node id="2" value="cek i<=2 "/>

<node id="6" value="menampilkan hasil" />

<node id="7" value="membaca inputan string" />

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int bil = 0;
            System.Console.Write("Bilangan = ");
            for (int i = 1; i <= 2; i++)
            {
                System.Console.Write("{0} ", i);
                bil += i;
            }
            System.Console.WriteLine();
            System.Console.ReadLine();
        }
     }
  }
```

Gambar 4.7 Kode Sumber Program Perulangan Menampilkan Bilangan 1 sampai 2

Pengujian pada penelitian dirancang dengan tujuan membandingkan struktur kode program dengan kasus uji yang berhasil dibentuk. Hasil yang ingin dicapai adalah untuk mengetahui konsistensi antara *Program Execution Traces* (PET) saat kode program dijalankan dengan kasus uji yang berhasil dibentuk.

Program Execution Traces (PET) dilakukan terhadap kode sumber pada Gambar 4.7 untuk kasus uji yang menunjukkan perulangan dengan menampilkan bilangan 1-2 dengan data uji yang sudah ditentukan, hasilnya adalah sebagai berikut.

Step 1 -> int bil = 0;
Step 2 -> System.Console.Write ("Bilangan = ");
Step 3 -> int i = 1;
Step 4 -> i <= 2;
Step 5 -> System.Console.Write("{0} ", i);
Step 6 -> bil += i;
Step 7 -> i++;
Step 8 -> i <= 2;
Step 9 -> System.Console.Write("{0} ", i);
Step 10 -> bil += i;
Step 11 -> i++;
Step 12 -> i <= 2;
Step 13 -> System.Console.WriteLine();
Step 14 -> System.Console.ReadLine();

Program Execution Traces (PET) dilakukan terhadap kode sumber pada Gambar 4.7 untuk kasus uji yang menunjukkan perulangan yang sudah dilakukan lebih dari 2 dengan masukan tidak valid yang sudah ditentukan, hasilnya adalah sebagai berikut.

Step 1 -> int bil = 0;
Step 2 -> System.Console.Write ("Bilangan = ");
Step 3 -> int i = 1;
Step 4 -> i <= 0;
Step 5 -> System.Console.WriteLine();
Step 6 -> System.Console.ReadLine();

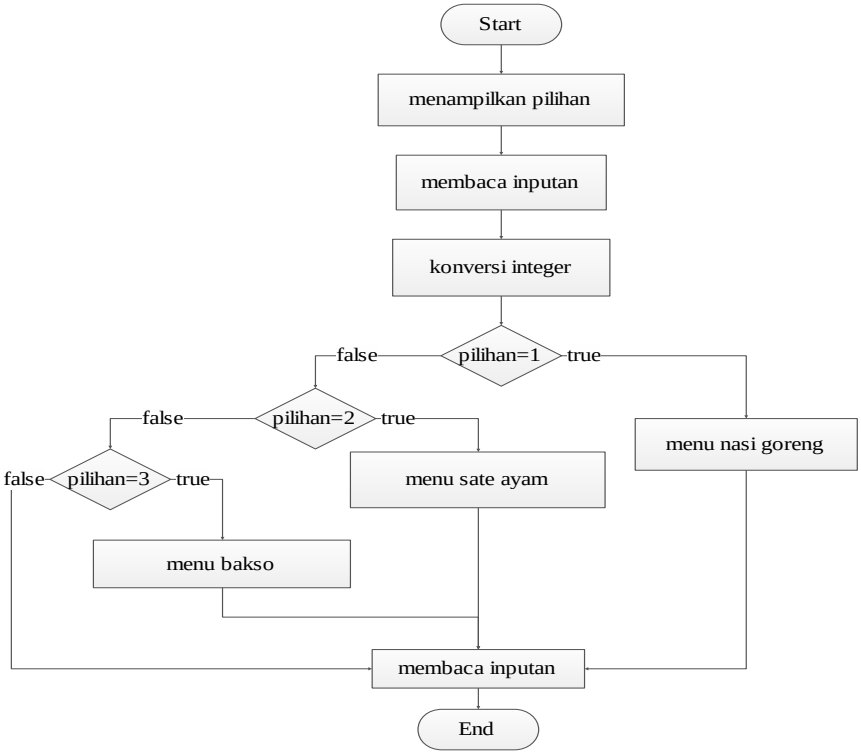
Tabel 4.3 Hasil Pengujian Terhadap Diagram Alir 2

No		Masukan	Keluaran	Kasus Uji
1	VALID	2	1,2	Tidak Konsisten
2	VALID	8	1,2,3,4,5,6,7,8	Tidak Konsisten
3	TIDAK VALID	0	null	Konsisten
4	TIDAK VALID	-2	null	Konsisten

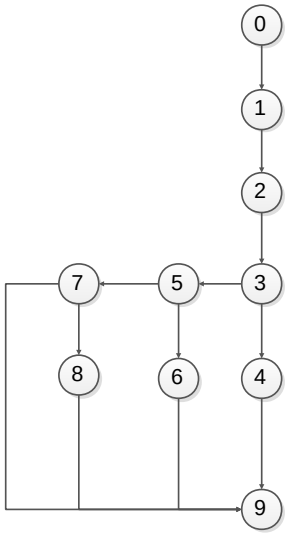
Dari hasil pengujian contoh studi kasus terhadap diagram alir 2 didapatkan hasil seperti yang sudah dijabarkan pada Tabel 4.3. Dapat dilihat dari tahapan pengujian yang dilakukan tersebut bahwa ketika data uji yang dimasukkan adalah valid, kasus uji dengan *Program Execution Traces* (PET) tidak sesuai. Hal ini dikarenakan pencarian semua kemungkinan kasus uji menggunakan strategi DFS, sehingga tidak terjadi perulangan pada suatu node yang sudah dilewati. Artinya, bahwa kasus uji yang dihasilkan secara otomatis tidak konsisten terhadap *Program Execution Traces* (PET). Akan tetapi, ketika data uji dimasukkan adalah tidak valid, maka kasus uji dengan *Program Execution Traces* (PET) adalah sesuai. Hal ini dikarenakan ketika data uji tidak valid, alur yang dituju tidak mengandung perulangan. Hal ini dapat diartikan bahwa kasus uji yang dihasilkan secara otomatis konsisten terhadap *Program Execution Traces* (PET).

4.2.3.7. Pembentukan Kasus Uji 3

Gambar 4.8 merupakan diagram alir dari kode sumber program merepresentasikan bentuk umum dari diagram alir yang mengandung percabangan switch case. Kasus uji dibentuk dari hasil konversi diagram alir ke dalam bentuk graf. Untuk dapat dikonversi ke dalam bentuk graf, maka diagram alir dimodelkan ke dalam bentuk metadata yang bertipe GML (*Geography Markup Language*). Langkah konversi ini dilakukan menggunakan kakas bantu notepad++. Pada tahap ini, graf yang berhasil dibentuk akan diproses untuk menghasilkan semua kemungkinan kasus uji beserta dengan ketiga nilai cakupannya, yaitu *Node Coverage, Transition Coverage* dan *Simple Path Coverage*.



Gambar 4.8 Diagram Alir Menentukan Menu Makanan



Gambar 4.9 Flow Graph Menentukan Menu Makanan

4.2.3.8. Menghitung Kriteria Cakupan dari Kasus Uji 3

Dalam penelitian ini sudah ditentukan 3 (tiga) kriteria cakupan yaitu *Node Coverage*, *Transition Coverage* dan *Simple Path Coverage*. Berikut adalah penjelasan dan langkah-langkah yang dilakukan untuk menghitung kasus uji yang sesuai dengan kriteria cakupan.

Node Coverage adalah perhitungan nilai cakupan berdasarkan jumlah node yang terdapat pada kasus uji. *Transition Coverage* adalah perhitungan nilai cakupan berdasarkan jumlah transisi yang terdapat pada kasus uji. *Simple Path Coverage*

adalah perhitungan nilai cakupan berdasarkan jumlah alur uji yang tidak mengandung perulangan yang terdapat pada graf.

Graf yang sudah terbentuk dapat dilihat pada Gambar 4.9. Setelah graf terbentuk, maka langkah selanjutnya adalah menghitung berapa jumlah node, jumlah transisi dan jumlah *simple path* pada graf tersebut. Jumlah node adalah 10, jumlah transisi adalah 12, jumlah kmungkinan semua kasus uji adalah 4 dan jumlah *simple path* adalah 4. Selanjutnya, dihitung nilai cakupannya sesuai dengan kriteria yang sudah ditentukan.

1. 0->1->2->3->4->9.
2. 0->1->2->3->5->6->9.
3. 0->1->2->3->5->7->8->9
4. 0->1->2->3->5->7->9

Berikut adalah penjelasan dan langkah-langkah yang dilakukan untuk menghitung kasus uji yang sesuai dengan kriteria cakupan.

1. Node Coverage

Nilai cakupan dari *Node Coverage* merupakan rasio node yang berhasil tercakup dengan keseluruhan jumlah node pada graf dikalikan 100%.

0->1->2->3->5->7->9. Node yang tercakup adalah 7. Jumlah keseluruhan node adalah 10. Nilai cakupan = $\frac{7}{10}$ x 100% = 70%

2. Transition Coverage

Nilai cakupan dari *Transition Coverage* merupakan rasio transisi yang berhasil tercakup dengan keseluruhan jumlah transisi pada graf.

0->1->2->3->5->7->9. Transisi yang tercakup adalah 6. Jumlah keseluruhan transisi adalah 12. Nilai cakupan = $\frac{6}{12}$ x 100% = 50%

3. Simple Path Coverage

Nilai cakupan dari *Simple Path* yang tidak mengandung perulangan dan merupakan rasio simple path yang berhasil tercakup dengan keseluruhan *simple path* pada graf. Dalam contoh kasus yang diberikan, simple path yang terbentuk adalah 4. Dan semua kemungkinan kasus uji yang diberikan pada contoh studi kasus adalah 4 simple path.

Nilai cakupan = $\frac{2}{2}$ x 100% = 100%.

4.2.3.9. Pengujian Konsistensi Antara PET dengan Kasus Uji 3

Berdasarkan kasus uji yang berhasil dibentuk secara otomatis dari Gambar 4.10 adalah sejumlah 4, yaitu kasus uji dengan urutan langkah 0->1->2->3->4->9 yang menunjukkan menu pilihan 1. Kasus uji dengan urutan 0->1->2->3->5->6->9 yang menunjukkan menu pilihan 2. Kasus uji dengan urutan 0->1->2->3->5->7->8->9 yang menunjukkan menu pilihan 3. Kasus uji dengan urutan 0->1->2->3->5->7->9 yang menunjukkan tidak memilih menu makanan 1,2 ataupun 3.

Untuk urutan langkah kasus uji yang menunjukkan menu pilihan 1 adalah sebagai berikut.

<node id="0" value="menampilkan pilihan;"/>
<node id="1" value="membaca inputan"/>
<node id="2" value="konversi integer"/>
<node id="3" value="pilihan=1"/>
<node id="4" value="menu nasi goreng"/>
<node id="9" value="membaca inputan string"/>

Untuk urutan langkah kasus uji yang menunjukkan menu pilihan 2 adalah sebagai berikut.

<node id="0" value="menampilkan pilihan;"/>
<node id="1" value="membaca inputan"/>
<node id="2" value="konversi integer"/>
<node id="3" value="pilihan=1"/>
<node id="5" value="pilihan=2"/>
<node id="6" value="menu sate ayam"/>
<node id="9" value="membaca inputan string"/>

Untuk urutan langkah kasus uji yang menunjukkan menu pilihan 3 adalah sebagai berikut.

<node id="0" value="menampilkan pilihan;"/>
<node id="1" value="membaca inputan"/>
<node id="2" value="konversi integer"/>
<node id="3" value="pilihan=1"/>
<node id="5" value="pilihan=2"/>
<node id="7" value="pilihan=3"/>
<node id="8" value="menu bakso"/>
<node id="9" value="membaca inputan string"/>

Untuk langkah kasus uji yang menunjukkan tidak memilih menu makanan 1,2 ataupun 3 adalah sebagai berikut.

<node id="0" value="menampilkan pilihan;"/>

<node id="1" value="membaca inputan"/>

<node id="2" value="konversi integer"/>

<node id="3" value="pilihan=1"/>

<node id="5" value="pilihan=2"/>

<node id="7" value="pilihan=3"/>

<node id="9" value="membaca inputan string"/>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Masukkan pilihan anda (1-3: );
            string pilihan = Console.ReadLine();
            int i = Convert.ToInt16(pilihan);
            switch (i)
            {
                case 1:
                  Console.WriteLine("Anda memilih nasi goreng");
                  break;
                case 2:
                  Console.WriteLine("Anda Memilih sate ayam");
                  break;
                case 3:
                  Console.WriteLine("Anda memilih bakso");
                    break;
            }
            Console.ReadLine();
        }
    }
}
```

Gambar 4.10 Kode Sumber Program Console Menentukan Menu Makanan

Selanjutnya dilakukan *Program Execution Traces* (PET) terhadap program untuk melacak setiap langkah yang dilalui ketika data uji dimasukkan. Tujuannya adalah untuk mengetahui konsistensi alur kasus uji terhadap *Program Execution Traces* (PET). *Program Execution Traces* (PET) dilakukan terhadap kode sumber pada Gambar 4.10

Program Execution Traces (PET) dilakukan untuk kasus uji yang menunjukkan menu pilihan 1 dengan data uji yang sudah ditentukan, hasilnya adalah sebagai berikut.

Step 1 -> Console.Write("Masukkan pilihan anda (1-3): ");

Step 2 -> string pilihan = Console.ReadLine();

Step 3 -> int i = Convert.ToInt16(pilihan);

Step 4 -> switch (i)

Step 5 -> Console.WriteLine("Anda memilih nasi goreng");

Step 6 -> break;

Step 7 -> System.Console.ReadLine();

Program Execution Traces (PET) dilakukan untuk kasus uji yang menunjukkan menu pilihan 2 dengan data uji yang sudah ditentukan, hasilnya adalah sebagai berikut.

Step 1 -> Console.Write("Masukkan pilihan anda (1-3): ");

Step 2 -> string pilihan = Console.ReadLine();

Step 3 -> int i = Convert.ToInt16(pilihan);

Step 4 -> switch (i)

Step 5 -> Console.WriteLine("Anda Memilih sate ayam");

Step 6 -> break;

Step 7 -> System.Console.ReadLine();

Program Execution Traces (PET) dilakukan untuk kasus uji yang menunjukkan menu pilihan 3 dengan data uji yang sudah ditentukan, hasilnya adalah sebagai berikut.

Step 1 -> Console.Write("Masukkan pilihan anda (1-3): ");

Step 2 -> string pilihan = Console.ReadLine();

Step 3 -> int i = Convert.ToInt16(pilihan);

Step 4 -> switch (i)

Step 5 -> Console.WriteLine("Anda memilih bakso");

Step 6 -> break;

Step 7 -> System.Console.ReadLine();

Program Execution Traces (PET) dilakukan untuk kasus uji yang menunjukkan tidak memilih menu makanan 1,2 ataupun 3 dengan data uji yang sudah ditentukan, hasilnya adalah sebagai berikut.

Step 1 -> Console.Write("Masukkan pilihan anda (1-3): ");

Step 2 -> string pilihan = Console.ReadLine();

Step 3 -> int i = Convert.ToInt16(pilihan);

Step 4 -> switch (i)

Step 5 -> System.Console.ReadLine();

Tabel 4.4 Hasil Pengujian Terhadap Diagram Alir 3

No		Masukan	Keluaran	Kasus Uji
1	VALID	1	Nasi goreng	Tidak Konsisten
2	VALID	2	Sate Ayam	Tidak Konsisten
3	VALID	3	Bakso	Tidak Konsisten
4	TIDAK VALID	4	null	Tidak Konsisten

Dari hasil pengujian contoh studi kasus terhadap diagram alir 3 didapatkan hasil seperti yang sudah dijabarkan pada Tabel 4.4. Dapat dilihat dari tahapan pengujian yang dilakukan tersebut bahwa ketika data uji yang dimasukkan valid dan tidak valid, kasus uji dengan *Program Execution Traces* (PET) adalah tidak sesuai. Hal ini terjadi karena kasus uji dibentuk berdasarkan diagram alir, dimana pada setiap kondisi percabangan, kondisi tersebut dianggap node. Sementara pada PET yang dilakukan, percabangan yang menggunakan switch case tidak mencakup kondisi percabangan. Sehingga, kasus uji yang dibentuk berdasarkan diagram alir tidak sesuai terhadap PET.

4.3. Analisis Hasil

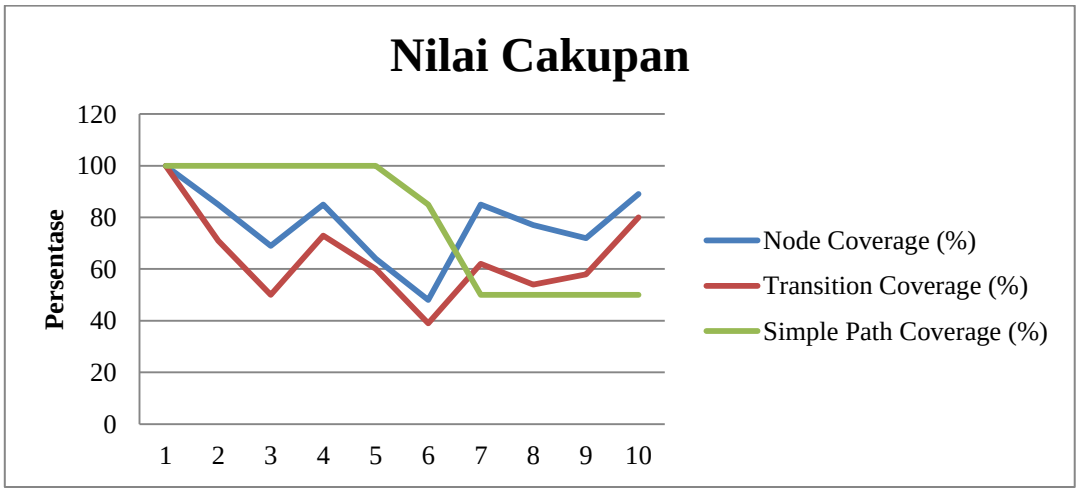
Dalam penelitian ini dilakukan pengujian untuk mengetahui konsistensi antara kasus uji yang dibentuk secara otomatis berdasarkan diagram alir dengan *Program Execution Traces* (PET). Setiap diagram alir yang digunakan sebagai masukan dapat menghasilkan lebih dari 1(satu) kasus uji. Berikut pada Tabel 4.3 dijabarkan hasil perhitungan cakupan pada setiap diagram alir dan Gambar 4.5 merupakan gambar grafk dari nilai cakupan berdasarkan 3 kriteria cakupan yang sudah ditentukan.

Tabel 4.5 merupakan hasil uji coba dengan menggunakan 10 diagram alir dan berhasil membentuk 29 kasus uji. Berdasarkan hasil penelitian yang dijabarkan pada Tabel 4.5, dapat disimpulkan bahwa nilai *Node Coverage* akan semakin tinggi jika semakin banyak node yang tercakup. *Node Coverage* akan bernilai 100% jika semua node tercakup. Begitu juga dengan nilai pada *Transition Coverage* akan semakin tinggi jika semakin banyak transisi yang tercakup. *Transition Coverage* akan bernilai 100% jika semua transisi tercakup. *Simple Path*

Coverage akan bernilai 100% jika dalam kode sumber program tidak terdapat perulangan (*looping*). Apabila *Simple Path Coverage* bernilai kurang dari 100%, maka dalam kode sumber program tersebut terdapat perulangan (*looping*).

Tabel 4.5 Hasil Uji Coba Terhadap 10 Diagram Alir

Diagram Alir ke-	Kasus Uji	Node	Transisi	Simple Path	Node Coverage (%)	Transition Coverage (%)	Simple Path Coverage (%)
1	1	15	14	1	100	100	100
2	2	7	7	2	85	71	100
3	5	13	16	5	69	50	100
4	3	14	15	3	85	73	100
5	3	34	35	3	64	60	100
6	7	33	38	7	48	39	85
7	2	7	8	1	85	62	50
8	2	9	11	1	77	54	50
9	2	11	12	1	72	58	50
10	2	19	20	1	89	80	50



Gambar 4.11 Grafik NilaiCakupan Berdasarkan 3 Kriteria

Dalam penelitian ini dilakukan perhitungan akurasi terhadap kasus uji yang dibentuk. Tujuannya adalah untuk mengetahui apakah kasus uji konsisten terhadap *Program Execution Traces* (PET). Akurasi merupakan perbandingan jumlah kasus uji yang sesuai dengan kasus uji keseluruhan yang berhasil dibentuk. Untuk formula akurasi dapat dilihat pada persamaan 4.1.

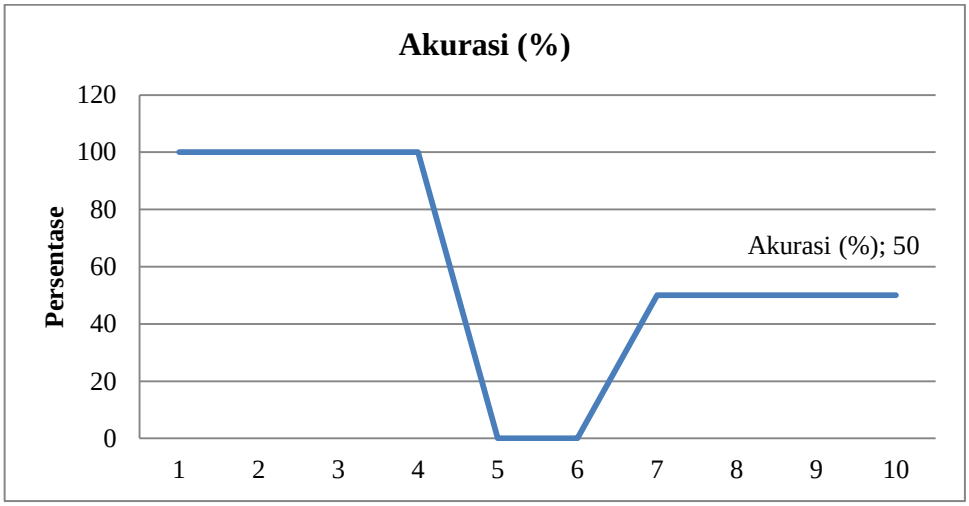
$$\text{Akurasi} = \frac{\text{jumlah kasus uji yang sesuai}}{\text{jumlah keseluruhan kasus uji}} \; x \; 100\% \qquad (4.1)$$

Untuk menilai akurasi hasil yang didapatkan, dilakukan pengujian secara manual yaitu dengan menjalankan kode sumber program, kemudian memasukkan data uji yang bernilai valid dan tidak valid. Untuk dapat mengetahui apakah instruksi yang terdapat pada program berjalan dan sesuai dengan pelabelan pada

node graf, dilakukan proses *debugging* setiap baris kode sumber program menggunakan IDE Microsoft Visual 2008. Untuk tabel perhitungan akurasi kesesuaian dan grafiknya dapat dilihat pada Tabel 4.4 dan Gambar 4.6.

Tabel 4.6 Hasil Perhitungan Akurasi Kesesuaian Kasus Uji terhadap PET

Diagram Alir ke-	Kasus uji yang terbentuk	Kasus uji yang sesuai	Akurasi (%)
1	1	1	100
2	2	2	100
3	5	5	100
4	3	3	100
5	3	0	0
6	7	0	0
7	2	1	50
8	2	1	50
9	2	1	50
10	2	1	50



Gambar 4.12 Grafik Nilai Akurasi dari Kesesuaian Kasus Uji terhadap PET

Tabel 4.6 menunjukkan hasil perhitungan akurasi kesesuaian kasus uji terhadap PET. Gambar 4.12 merupakan gambaran grafik persentase dari nilai akurasi kesesuaian. Seperti yang dijabarkan pada Tabel 4.6 dan Gambar 4.12 dapat disimpulkan bahwa dari semua kasus uji yang dibentuk secara otomatis berdasarkan diagram alir, tidak semuanya sesuai terhadap *Program Execution Traces* (PET). Kasus uji yang berhasil dibentuk sejumlah 29, yang sesuai sejumlah 15 sedangkan yang tidak sesuai sejumlah 14. Berikut adalah penjelasan rinci terhdap analisa yang dilakukan.

2. Pada diagram alir ke 1, kode sumber program tidak mengandung percabangan dan perulangan. Kasus uji yang terbentuk sesuai dengan PET yang diuji secara manual. Nilai akurasi yang didapatkan adalah 100%.
3. Pada diagram alir ke 2, 3, dan 4, kode sumber program mengandung percabangan *if else*. Kasus uji yang terbentuk sesuai dengan PET yang diuji secara manual. Nilai akurasi yang didapatkan adalah 100%.
4. Pada diagram alir ke 5 dan 6, kode sumber program mengandung percabangan switch case. Kasus uji yang terbentuk tidak sesuai dengan PET yang diuji secara manual. Nilai akurasi yang didapatkan adalah 0%. Hal ini terjadi karena kasus uji dibentuk berdasarkan diagram alir, dimana pada setiap kondisi percabangan, kondisi tersebut dianggap node. Sementara pada PET yang dilakukan, percabangan yang menggunakan switch case tidak mencakup kondisi percabangan. Sehingga, kasus uji yang dibentuk berdasarkan diagram alir tidak sesuai terhadap PET.
5. Pada diagram alir ke 7, 8, 9 dan 10, kode sumber program mengandung perulangan for dan while. Kasus uji yang terbentuk sebagian sesuai dengan PET yang diuji secara manual. Nilai akurasi yang didapatkan adalah 50%. Hal ini terjadi karena untuk mencari kemungkinan kasus uji menggunakan strategi DFS, sehingga tidak terjadi perulangan pada suatu node yang sudah dilewati. Ketika kasus uji yang dibentuk tidak mencakup perulangan, maka kasus uji tersebut sesuai.

BAB 5
KESIMPULAN DAN SARAN

Pada bab ini dijelaskan mengenai kesimpulan akhir yang didapat setelah melakukan serangkaian uji coba pada bab sebelumnya.

5.1. Kesimpulan

Kesimpulan yang dapat diambil dalam penelitian ini antara lain adalah sebagai berikut.

a. Proses konversi diagram alir ke dalam bentuk graf, dilakukan dengan memodelkan metadata diagram alir ke dalam format GML (*Geography Markup Language*). Langkah ini dilakukan secara manual menggunakan Notepad++.
b. Pembentukan kasus uji berdasarkan diagram alir menggunakan metode DFS dapat menghasilkan kasus uji dengan akurasi cukup, yaitu 60%. Pada penelitian ini, masukan yang digunakan adalah diagram alir dan keluarannya adalah kasus uji. Diagram alir yang digunakan sejumlah 10, dan kasus uji yang berhasil dibentuk sejumlah 29. Berdasarkan dari hasil uji coba dari 29 kasus uji yang berhasil dibentuk, kasus uji yang sesuai dengan *Program Execution Traces* (PET) sejumlah 15, dan yang tidak sesuai sejumlah 14. Hal ini dikarenakan PET yang mengandung percabangan switch case tidak mencakup kondisi percabangan. Selain itu, juga tidak terjadi perulangan pada suatu node yang sudah dilewati pada kode sumber yang mengandung *for* dan *while*.
c. Penelitian ini menggunakan 3 kriteria cakupan yang sudah ditentukan, yaitu *Node Coverage, Transition Coverage* dan *Simple Path Coverage*. Berdasarkan hasil uji coba didapatkan hasil rata-rata *Node Coverage* 77,4%, *Transition Coverage* 64,7%, dan *Simple Path Coverage* 78,5%. Hasil yang didapatkan tersebut dapat digunakan oleh tester untuk mengetahui kompleksitas tingkat pengujian. Jika pada perhitungan terhadap *Node Coverage* dan *Transition Coverage* semakin kecil, maka pengujian yang dilakukan semakin kompleks. Untuk *Simple Path Coverage* semakin besar hasil yang didapatkan maka kasus uji dan pengujiannya sederhana.
d. Kasus uji yang sesuai adalah kasus uji yang tidak mengandung percabangan dan perulangan, selain itu juga kasus uji yang mengandung percabangan menggunakan if else. Kasus uji yang tidak sesuai adalah kasus uji yang

mengandung percabangan menggunakan switch case. Sedangkan kasus uji yang mengandung perulangan, memiliki kesesuaian sebagian saja.

5.2. Saran

Saran yang dapat diberikan untuk pengembangan lebih lanjut penelitian ini adalah sebagai berikut.

a. Penelitian ini masih belum berhasil dalam menghasilkan kasus uji yang sesuai dengan kondisi percabangan *switch case* hingga mencapai hasil akurasi yang lebih baik. Permasalahan ini dapat diperbaiki pada tahapan konversi kode sumber program ke diagram alir. Sehingga, pada saat melakukan konversi diagram alir untuk percabangan *switch case*, konverter dapat menghasilkan diagram alir yang sesuai dengan PET kode sumber program yang mengandung percabangan *switch case*. Diharapkan untuk pengembangan selanjutnya, permasalahan ini dapat diselesaikan.
b. Penelitian ini masih belum berhasil dalam menghasilkan kasus uji yang sesuai dengan kondisi perulangan hingga mencapai hasil akurasi yang lebih baik. Permasalahan ini dapat diperbaiki pada tahapan pembentukan kasus uji. Pembentukan kasus uji pada penelitian ini menggunakan strategi DFS, sehingga tidak dapat menghasilkan perulangan *node*. Diharapkan untuk pengembangan selanjutnya, permasalahan ini dapat diselesaikan dengan menggunakan strategi pembentukan kasus uji yang lain.

DAFTAR PUSTAKA

A.V.K. Shanthi, G. M. (2012). Automated Test Cases Generation from UML Sequence Diagram. *International Conference on Software and Computer Applications (ICSCA), 7*.

Abhishek Singhal, S. C. (2012, May). A Novel Approach For Priortization Of Optimized Test Cases. *International Journal on Computer Science and Engineering (IJCSE), 4*.

Ajay Kumar Jena, S. K. (2014). A Novel Approach for Test Case Generation from UML Activity Diagram. *International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 9*.

Archana Tomar, P. S. (2016, April). Software Testing with Different Optimization Techniques. *International Journal of Emerging Technology and Advanced Engineering, 6*(4).

Baikuntha Narayan Biswal, S. S. (2010). A Novel Approach for Optimized Test Case Generation using Activity and Collaboration Diagram. *International Journal of Computer Applications, 1*(14).

Itti Hooda, P. R. (2015, February). Software Test Process, Testing Types and Techniques. *International Journal of Computer Applications , 0975 – 8887*, 5.

Mahesh Shirole, A. S. (2011). Generation of improved Test Case from UML State Diagram using Genetic Algorithm. *ACM, 125-134*, 10.

Mäkinen, M. A. (2007). *Model Based Approach to Software Testing.* Department of Electrical and Communications Engineering. Networking Laboratory, Helsinki University of Technology.

Manisha Verma, P. A. (2016). A Study on Test Case Generation. *RIEECE-2016, 3*(1).

Manoj Kumar, A. S. (2011, November). Optimization of Test Cases using Soft Computing Techniques: A Critical Review. *WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, 8*(11), 13.

Mingsong Chen, X. Q. (2007, August). UML Activity Diagram-Based Automatic Test Case Generation For Java Programs. *The Computer Journal Advance*.

Umi Proboyekti, S. M. (2013). Pemodelan Perangkat Lunak. Chapter 2 Flowchart. 41-44.

V.Mary Sumalatha, D. G. (2009). An Model Based Test Case Generation Technique Using Genetic Algorithms. *The International Journal of Computer Science & Applications (TIJCSA)*, 12.

V.Mary Sumalatha, G. P. (2013, January). Object Oriented Test Case Generation Technique using Genetic Algorithms. *International Journal of Computer Applications , 0975 – 8887,* 13.

LAMPIRAN

LAMPIRAN 1 Data Set yang Digunakan

1. Program Menghitung penjumlahan, pengurangan, perkalian, dan pembagian dari 2 buah variabel yang diinputkan

```
1.   using System;
2.   using System.Collections.Generic;
3.   using System.Linq;
4.   using System.Text;
5.
6.   namespace all
7.   {
8.       class Program
9.       {
10.          static void Main(string[] args)
11.          {
12.              int a,b,jum, kurg, kali;
13.              decimal bagi;
14.              System.Console.Write("Masukan bilangan I  : ");
15.              a = int.Parse(System.Console.In.ReadLine());
16.              System.Console.Write("Masukan bilangan II : ");
17.              b= int.Parse(System.Console.In.ReadLine());
18.              jum = a + b;
19.              kurg = a - b;
20.              kali = a * b;
21.              bagi = a / b;
22.              System.Console.WriteLine("Hasil penjumlahan
     "+jum);
23.              System.Console.WriteLine("Hasil pengurangan
     "+kurg);
24.              System.Console.WriteLine("Hasil perkalian "+kali);
25.              System.Console.WriteLine("Hasil pembagian "+bagi);
26.
27.              System.Console.ReadLine();
28.          }
29.      }
30.  }
```

2. Program Menentukan bilangan genap dan ganjil sebuah bilangan

```
1.   using System;
2.   using System.Collections.Generic;
3.   using System.Linq;
4.   using System.Text;
5.
6.   namespace all
7.   {
8.       class Program
9.       {
10.          static void Main(string[] args)
11.          {
12.               int bil;
13.               System.Console.Write("Input Bilangan = ");
14.               bil = int.Parse(System.Console.ReadLine());
15.
16.               if (bil % 2 == 0)
17.            System.Console.WriteLine("Bil " + bil + "adalah
      genap");
```

```
18.                   else
19.                       System.Console.WriteLine("Bil " + bil +
   " adalah ganjil");
20.                   System.Console.ReadLine();
21.               }
22.           }
23.       }
```

3. Program Menentukan predikat suatu nilai

```
1.   using System;
2.   using System.Collections.Generic;
3.   using System.Linq;
4.   using System.Text;
5.
6.   namespace all
7.   {
8.       class Program
9.       {
10.          static void Main(string[] args)
11.          {
12.               int nilai;
13.               System.Console.Write("Masukan Nilai Anda : ");
14.               nilai = int.Parse(System.Console.ReadLine());
15.               if ((nilai >= 80) && (nilai <= 100))
16.                     {
17.                   System.Console.Write("Nilai Anda A");}
18.               else if ((nilai >= 70) && (nilai <= 80))
19.                     {
20.                   System.Console.Write("Nilai Anda B");}
21.               else if ((nilai >= 60) && (nilai <= 70))
22.                     {
23.                   System.Console.Write("Nilai Anda C");}
24.               else if ((nilai >= 50) && (nilai <= 60))
25.                     {
26.                   System.Console.Write("Nilai Anda D");}
27.               else System.Console.Write("Nilai E");
28.               System.Console.ReadLine() ;
29.           }
30.       }
31.   }
```

4. Program Menentukan nilai terbesar dari 3 buah bilangan

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5.
6. namespace all
7. {
8.     class Program
9.     {
10.             static void Main(string[] args)
11.             {
12.                int a, b, c;
13.                     Console.Write("Masukkan Angka 1 : ");
14.                    a = Convert.ToInt32(Console.ReadLine());
```

```
15.                     Console.Write("Masukkan Angka 2 : ");
16.                     b = Convert.ToInt32(Console.ReadLine());
17.                     Console.Write("Masukkan Angka 3 : ");
18.                     c = Convert.ToInt32(Console.ReadLine());
19.                     Console.WriteLine("-------------------------
   ------");
20.
21.                     if (a > c & a > b)
22.                     {
23.                         System.Console.WriteLine(a);
24.                     }
25.                     else if (b > a & b > c)
26.                     {
27.                         System.Console.WriteLine(b);
28.                     }
29.                     else
30.                     {
31.                         System.Console.WriteLine(c);
32.                     }
33.                     System.Console.ReadLine();
34.           }
35.       }
36.    }
```

5. Program Menghitung luas persegi dan lingkaran

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5.
6. namespace all
7. {
8.     class Program
9.     {
10.             static void Main(string[] args)
11.             {
12.                     int pilih;
13.                     double luas, keliling;
14.                     string jenis;
15.                     double phi = 3.14;
16.
17.                     System.Console.WriteLine("=====MENCARI LUAS
   DAN KELILING=====");
18.                     System.Console.WriteLine("-> ketikan 1 untuk
   bujur sangkar");
19.                     System.Console.WriteLine("-> ketikan 2 untuk
   lingkaran");
20.                     System.Console.Write("PILIHAN KAMU (1/2)? ->
   ");
21.                     pilih = int.Parse(System.Console.ReadLine());
22.                     System.Console.WriteLine();
23.                     switch (pilih)
24.                     {
25.                         case 1:
26.                             int sisi;
27.                             jenis = "Bujur Sangkar";
```

```
28.                             System.Console.Write("Masukan panjang
   sisi nya = ");
29.                             sisi =
   int.Parse(System.Console.ReadLine());
30.                             luas = sisi * sisi;
31.                             keliling = 4 * sisi;
32.
33.                             System.Console.WriteLine();
34.                             System.Console.WriteLine("Anda
   memilih " + jenis);
35.                             System.Console.WriteLine("Luas " +
   jenis + " adalah " + luas);
36.                             System.Console.WriteLine("Keliling "
   + jenis + " adalah " + keliling);
37.
38.                            break;
39.
40.                         case 2:
41.                             double jari2;
42.                             jenis = "Lingkaran";
43.                             System.Console.Write("Masukan jari-
   jari lingkaran = ");
44.                             jari2 =
   double.Parse(System.Console.ReadLine());
45.                             luas = 2 * phi * jari2;
46.                             keliling = phi * jari2 * jari2;
47.
48.                             System.Console.WriteLine();
49.                             System.Console.WriteLine("Anda
   memilih " + jenis);
50.                             System.Console.WriteLine("Luas " +
   jenis + " adalah " + luas);
51.                             System.Console.WriteLine("Keliling "
   + jenis + " adalah " + keliling);
52.
53.                            break;
54.
55.                         default:
56.                             System.Console.WriteLine("Tidak ada
   pilihan selain 1/2");
57.                             break;
58.                     }
59.
60.                     System.Console.ReadLine();
61.                 }
62.             }
63.         }
```

6. Program ATM

```
1. using System.Collections.Generic;
2. using System.Linq;
3. using System.Text;
4.
5. namespace all
6. {
7.     class Program
8.     {
9.         static void Main(string[] args)
10.        {
11.            int amount = 1000, deposit, withdraw;
12.            int choice, pin = 0;
13.
14.            Console.WriteLine("Enter Your Pin Number ");
15.            pin = int.Parse(Console.ReadLine());
16.            Console.WriteLine("********Welcome to ATM
  Service**************\n");
17.            Console.WriteLine("1. Check Balance\n");
18.            Console.WriteLine("2. Withdraw Cash\n");
19.            Console.WriteLine("3. Deposit Cash\n");
20.            Console.WriteLine("4. Quit\n");
21.            Console.WriteLine("**********************************
  ***********\n\n");
22.            Console.WriteLine("Enter your choice: ");
23.
24.            choice = int.Parse(Console.ReadLine());
25.            System.Console.WriteLine();
26.            switch (choice)
27.            {
28.              case 1:
29.              Console.WriteLine("\n YOUR BALANCE IN Rs : {0} ",
  amount);
30.              break;
31.              case 2:
32.              Console.WriteLine("\n ENTER THE AMOUNT TO WITHDRAW:
  ");
33.              withdraw = int.Parse(Console.ReadLine());
34.              if (withdraw % 100 != 0)
35.              {
36.               Console.WriteLine("\n PLEASE ENTER THE AMOUNT IN
  MULTIPLES OF 100");
37.               }
38.              else if (withdraw > (amount - 500))
39.               {
40.               Console.WriteLine("\n INSUFFICENT BALANCE");
41.              }
42.    else
43.              {
44.                 amount = amount - withdraw;
45.                 Console.WriteLine("\n\n PLEASE COLLECT CASH");
46.                 Console.WriteLine("\n YOUR CURRENT BALANCE IS
       {0}", amount);
47.                  }
```

```
48.        break;
49.                    case 3:
50.                    Console.WriteLine("\n ENTER THE AMOUNT TO
   DEPOSIT");
51.                    deposit = int.Parse(Console.ReadLine());
52.                    amount = amount + deposit;
53.                    Console.WriteLine("YOUR BALANCE IS {0}",
   amount);
54.                    break;
55.                    case 4:
56.                    Console.WriteLine("\n\n THANKS FOR USING
   OUT ATM SERVICE");
57.                    break;
58.                      }
59.                      System.Console.ReadLine();
60.                  }
61.              }
62.        }
```

7. Program Perulangan menampilkan bilangan dari 1 sampai 10

```
1. using System.Collections.Generic;
2. using System.Linq;
3. using System.Text;
4.
5. namespace all
6. {
7.     class Program
8.     {
9.         static void Main(string[] args)
10.              {
11.                  int bil = 0;
12.                  System.Console.Write("Bilangan = ");
13.                  for (int i = 1; i <= 10; i++)
14.                  {
15.                      System.Console.Write("{0} ", i);
16.                      bil += i;
17.                  }
18.                  System.Console.WriteLine();
19.                  System.Console.ReadLine();
20.              }
21.          }
22.      }
```

8. Program Perulangan Menampilkan bilangan genap dari 1 sampai 10

```
1. using System.Collections.Generic;
2. using System.Linq;
3. using System.Text;
4.
5. namespace all
6. {
7.     class Program
8.     {
9.         static void Main(string[] args)
10.              {
```

```
11.             int bil = 0;
12.             System.Console.Write("Bilangan genap = ");
13.
14.                 for (int i=1;i<=10;i++)
15.                 {
16.                     if (i %2==0)
17.                     {
18.                         System.Console.Write("{0} ",i);
19.                         bil+=i;
20.                     }
21.                 }
22.                     System.Console.WriteLine();
23.                     System.Console.WriteLine("Jumlah
    bilangan genap s/d 10   = " +bil);
24.                     System.Console.ReadLine();
25.         }
26.     }
27. }
```

9. Program Menghitung jumlah bilangan yang diinputkan

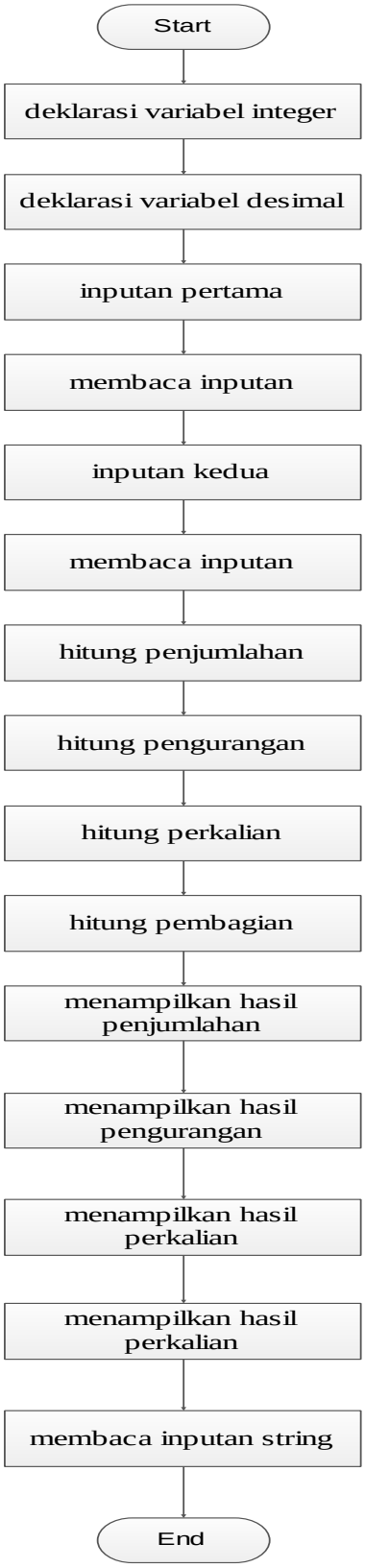
```
1.  using System.Collections.Generic;
2.  using System.Linq;
3.  using System.Text;
4.
5.  namespace all
6.  {
7.      class Program
8.      {
9.          static void Main(string[] args)
10.         {
11.             int i=1,n=0,jumlah=0,bil;
12.             System.Console.Write("Masukan banyaknya bilangan:
    ");
13.             n = int.Parse(System.Console.In.ReadLine());
14.
15.             while (i<=n)
16.             {
17.                 System.Console.Write("Masukan bilangan ke-"+i+"
    = " );
18.                 bil = int.Parse(System.Console.In.ReadLine());
19.                 jumlah = jumlah + bil;
20.                 i++;
21.             }
22.
23.             System.Console.WriteLine();
24.             System.Console.WriteLine("Hasil penjumlahan bilangan
    yang Anda masukan adalah = " +jumlah);
25.             System.Console.ReadLine();
26.         }
27.     }
28. }
```

10. Program Menghitung nilai rata- rata

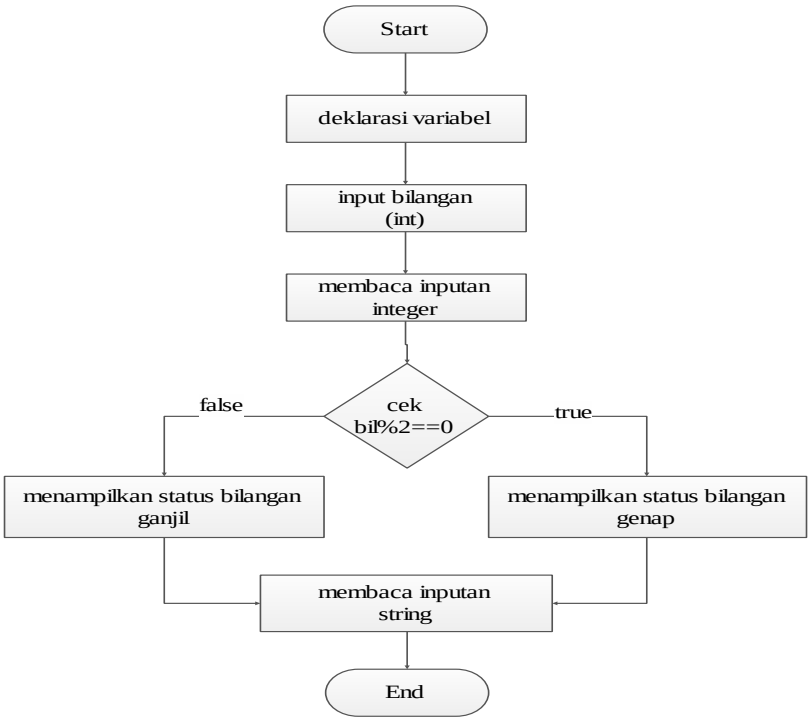
```
1.  using System.Collections.Generic;
2.  using System.Linq;
3.  using System.Text;
4.
5.  namespace all
6.  {
7.      class Program
8.      {
9.          static void Main(string[] args)
10.         {
11.             int[] nilai = new int[10];
12.                 int a, total, rata;
13.                 nilai[0] = 80;
14.                 nilai[1] = 90;
15.                 nilai[2] = 87;
16.                 nilai[3] = 85;
17.                 nilai[4] = 78;
18.
19.                 a = 0;
20.                 rata = 0;
21.                 total = 0;
22.                 Console.WriteLine("Berikut nilai siswa yang
    tersedia: ");
23.                 while(a < 5)
24.                 {
25.                     Console.WriteLine(" " + nilai[a]);
26.                     total = total + nilai[a];
27.                     a++;
28.                 }
29.                 Console.WriteLine(" Total nilai siswa nya
    adalah: " + total);
30.                 rata = total / a;
31.                 Console.WriteLine("Rata-rata nya adalah: " +
    rata);
32.                 System.Console.ReadLine();
33.         }
34.     }
35. }
```

LAMPIRAN 2 Diagram Alir

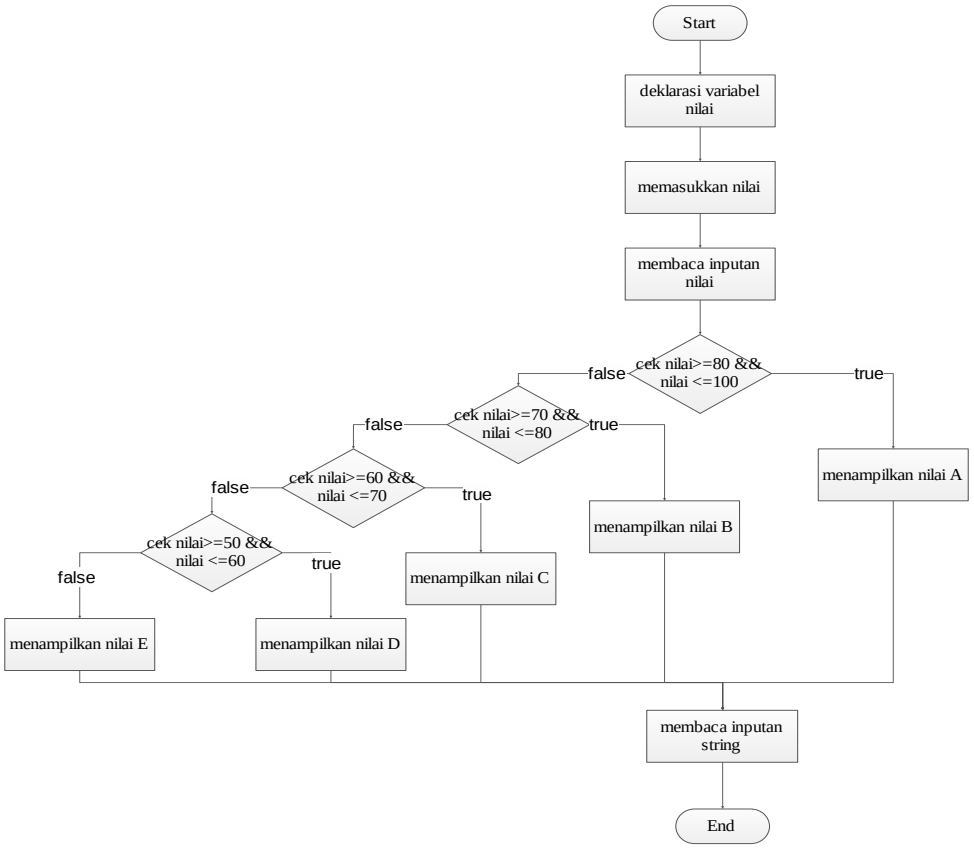
1. Diagram alir program menghitung penjumlahan, pengurangan, perkalian, dan pembagian dari 2 buah variabel yang diinputkan



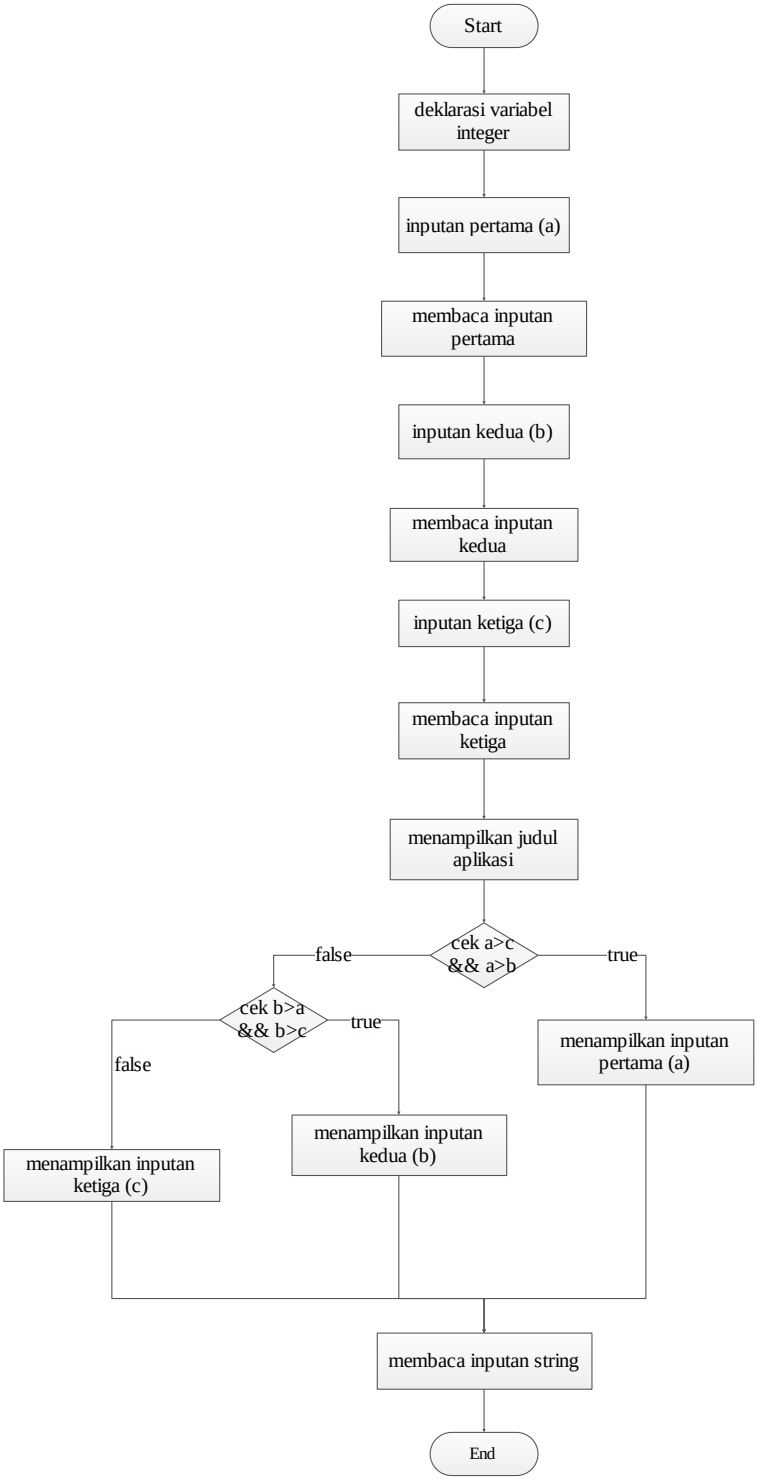
2. Diagram alir program menentukan bilangan genap dan ganjil sebuah bilangan



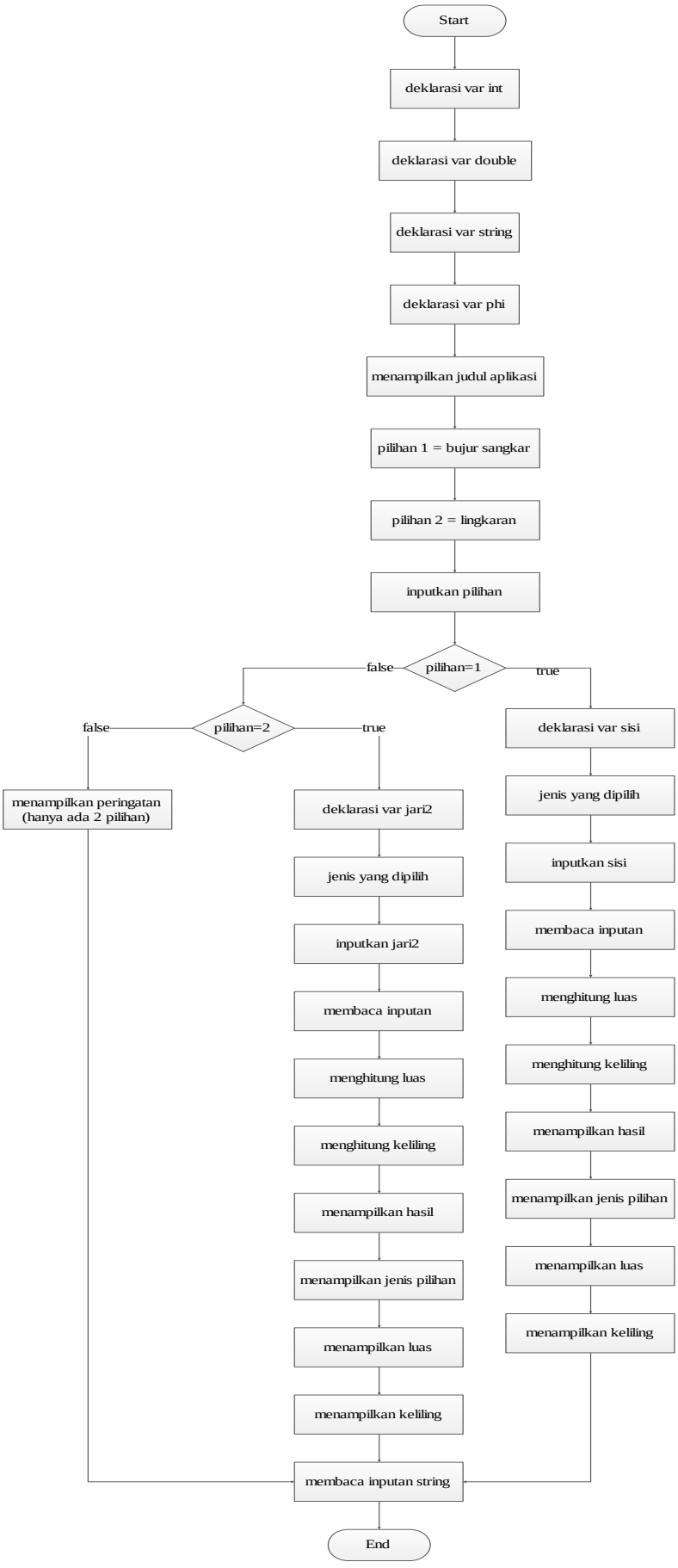
3. Diagram alir program menentukan predikat suatu nilai



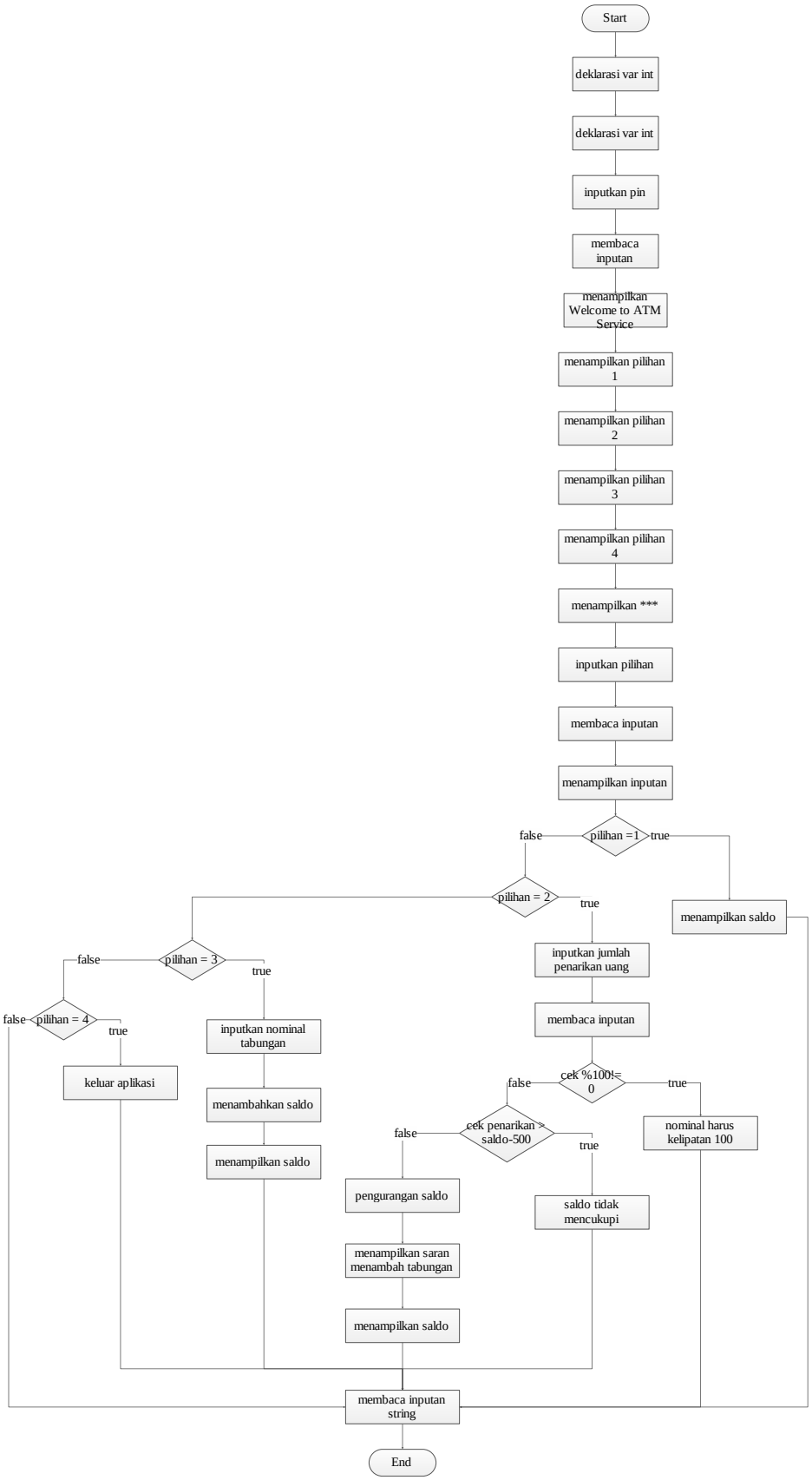
4. Diagram alir program menentukan nilai terbesar dari 3 buah bilangan



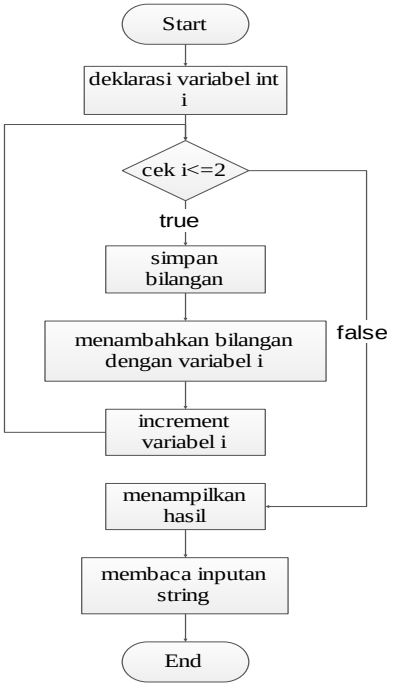
5. Diagram alir program menghitung luas persegi dan lingkaran



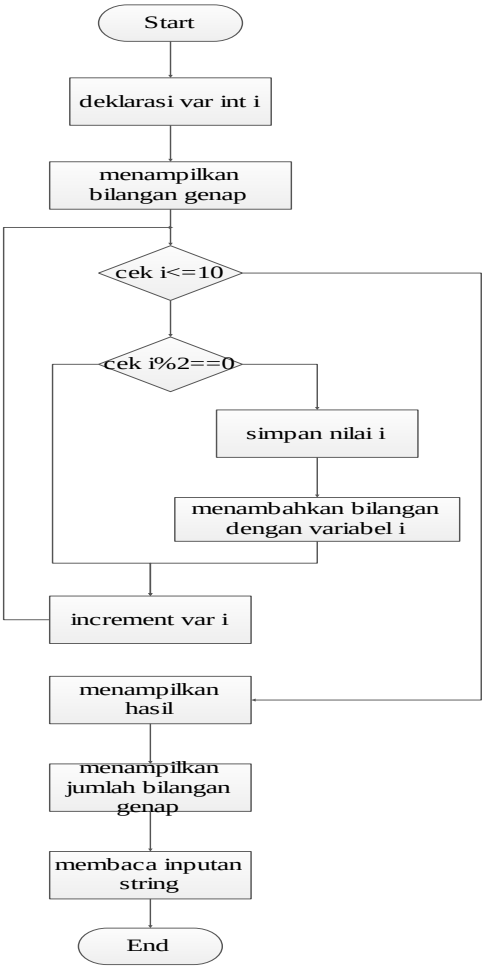
6. Diagram alir program ATM



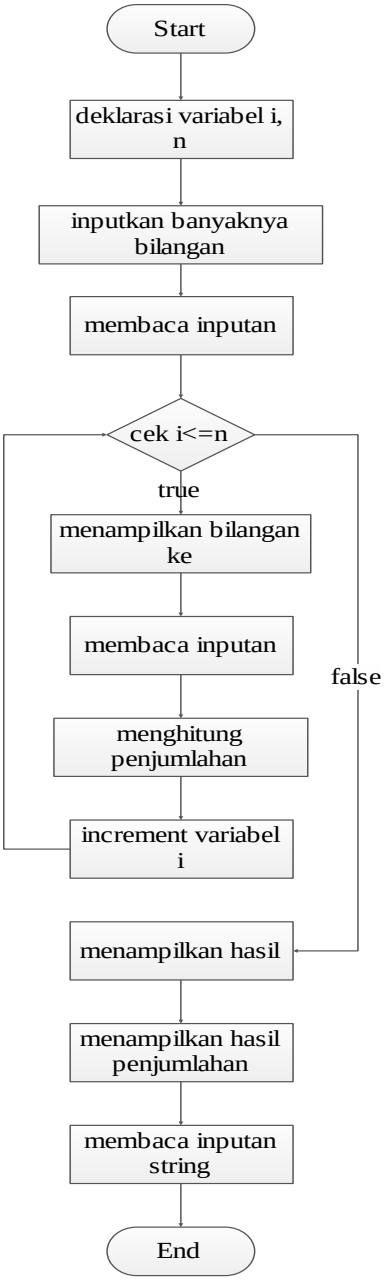
7. Diagram alir program perulangan menampilkan bilangan dari 1 sampai 10



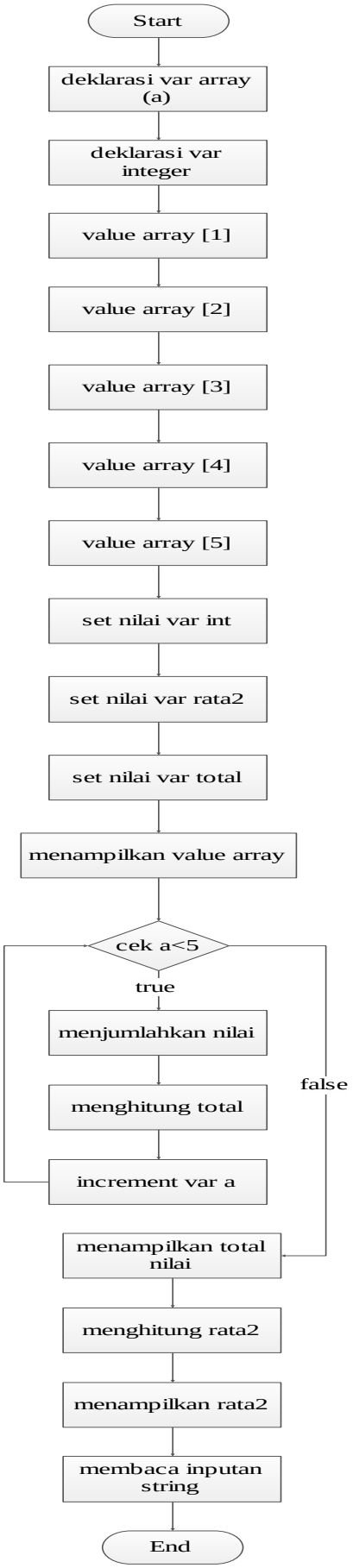
8. Diagram alir program perulangan menampilkan bilangan genap dari 1 sampai 10



9. Diagram alir program menghitung jumlah bilangan yang diinputkan



10. Diagram alir program menghitung nilai rata- rata



LAMPIRAN 3 GML (Geography Markup Language)

1. GML program menghitung penjumlahan, pengurangan, perkalian, dan pembagian dari 2 buah variabel yang diinputkan

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
<graph id="G" edgedefault="directed" parse.nodes="15"
parse.edges="14" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
    <nodes>
            <node id="0" value="deklarasi variabel integer"/>
            <node id="1" value="deklarasi variabel desimal
            <node id="2" value="inputan pertama"/>
            <node id="3" value="membaca inputan"/>
            <node id="4" value="inputan kedua"/>
            <node id="5" value="membaca inputan"/>
            <node id="6" value="rumus penjumlahan"/>
            <node id="7" value="rumus pengurangan"/>
            <node id="8" value="rumus perkalian"/>
            <node id="9" value="rumus pembagian"/>
            <node id="10" value="menampilkan hasil penjumlahan"/>
            <node id="11" value="menampilkan hasil pengurangan"/>
            <node id="12" value="menampilkan hasil perkalian"/>
            <node id="13" value="menampilkan hasil pembagian"/>
            <node id="14" value="membaca inputan string"/>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="3to4" source="3" target="4" />
            <edge id="4to5" source="4" target="5" />
            <edge id="5to6" source="5" target="6" />
            <edge id="6to7" source="6" target="7" />
            <edge id="7to8" source="7" target="8" />
            <edge id="8to9" source="8" target="9" />
            <edge id="9to10" source="9" target="10" />
            <edge id="10to11" source="10" target="11" />
            <edge id="11to12" source="11" target="12" />
            <edge id="12to13" source="12" target="13" />
            <edge id="13to14" source="13" target="14" />
      </edges>
  </graph>
</graphml>
```

2. GML program menentukan bilangan genap dan ganjil sebuah bilangan

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed" parse.nodes="7"
parse.edges="7" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
      <nodes>
            <node id="0" value="deklarasi variabel"/>
            <node id="1" value="input bilangan (int)"/>
            <node id="2" value="membaca inputan integer"/>
            <node id="3" value="cek bil % 2 == 0"/>
            <node id="4" value="menampilkan status bilangan
            ganjil);"/>
```

```
            <node id="5" value="menampilkan status bilangan
            genap);"/>
            <node id="6" value="membaca inputan string"/>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="3to4" source="3" target="4" />
            <edge id="4to6" source="4" target="6" />
            <edge id="3to5" source="3" target="5" />
            <edge id="5to6" source="5" target="6" />
      </edges>
  </graph>
</graphml>
```

3. GML program menentukan predikat suatu nilai

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
<graph id="G" edgedefault="directed" parse.nodes="13"
parse.edges="16" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
      <nodes>
            <node id="0" value="deklarasi variabel nilai "/>
            <node id="1" value="memasukkan nilai "/>
            <node id="2" value="membca inputan nilai"/>
            <node id="3" value="cek (nilai >=80)dan(nilai <=
            100)"/>
            <node id="4" value="menampilkan nilai A"/>
            <node id="5" value="jika (nilai >= 70) dan (nilai <=
            80)"/>
            <node id="6" value="menampilkan nilai B);"/>
            <node id="7" value="jika (nilai >= 60) dan (nilai <=
            70)"/>
            <node id="8" value="menampilkan nilai C"/>
            <node id="9" value="jika (nilai >= 50) dan (nilai <=
            60))"/>
            <node id="10" value="menampilkan nilai D"/>
            <node id="11" value="jika <50, menampilkan nilai E"/>
            <node id="12" value="membaca inputan string"/>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="3to4" source="3" target="4" />
            <edge id="4to12" source="4" target="12" />
            <edge id="3to5" source="3" target="5" />
            <edge id="5to6" source="5" target="6" />
            <edge id="6to12" source="6" target="12" />
            <edge id="5to7" source="5" target="7" />
            <edge id="7to8" source="7" target="8" />
            <edge id="8to12" source="8" target="12" />
            <edge id="7to9" source="7" target="9" />
            <edge id="9to10" source="9" target="10" />
            <edge id="10to12" source="10" target="12" />
            <edge id="9to11" source="9" target="11" />
            <edge id="11to12" source="11" target="12" />
      </edges>
</graph>
</graphml>
```

4. GML program menentukan nilai terbesar dari 3 buah bilangan

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
<graph id="G" edgedefault="directed" parse.nodes="14"
parse.edges="15" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
    <nodes>
            <node id="0" value="deklarasi variabel integer" />
            <node id="1" value="inputan pertama (a)"/>
            <node id="2" value="membaca  inputan pertama"/>
            <node id="3" value="inputan kedua (b)"/>
            <node id="4" value="membaca inputan kedua"/>
            <node id="5" value="inputan ketiga"/>
            <node id="6" value="membaca inputan ketiga (c)"/>
            <node id="7" value="menampilkan garis batas"/>
            <node id="8" value="cek a>c dan a>b)"/>
            <node id="9" value="menampilkan inputan pertama(a)" />
            <node id="10" value="cek jika b>a dan b>c" />
            <node id="11" value="menampilkan inputan kedua (b)"/>
            <node id="12" value="menampilkan inputan ketiga (c)"/>
            <node id="13" value="membaca inputan string" />
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="3to4" source="3" target="4" />
            <edge id="4to5" source="4" target="5" />
            <edge id="5to6" source="5" target="6" />
            <edge id="6to7" source="6" target="7" />
            <edge id="7to8" source="7" target="8" />
            <edge id="8to9" source="8" target="9" />
            <edge id="9to13" source="9" target="13" />
            <edge id="8to10" source="8" target="10" />
            <edge id="10to11" source="10" target="11" />
            <edge id="11to13" source="11" target="13" />
            <edge id="10to12" source="10" target="12" />
            <edge id="12to13" source="12" target="13" />
      </edges>
  </graph>
</graphml>
```

5. GML program menghitung luas persegi dan lingkaran

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed" parse.nodes="34"
parse.edges="35" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
    <nodes>
            <node id="0" value="deklarasi variabel integer" />
            <node id="1" value="deklarasi variabel double"/>
            <node id="2" value="deklarasi variabel string"/>
            <node id="3" value="deklarasi variabel phi"/>
            <node id="4" value="menampilkan judul aplikasi"/>
            <node id="5" value="pilihan 1 = bujur sangkar);" />
            <node id="6" value="pilihan 2 = lingkaran);"/>
            <node id="7" value="inputkan pilihan"/>
            <node id="8" value="membaca inputan"/>
            <node id="9" value="menampilkan inputan" />
            <node id="10" value="pilihan=1" />
```

```
            <node id="11" value="deklarasi var sisi"/>
            <node id="12" value="jenis yang dipilih"/>
            <node id="13" value="inputkan sisi"/>
            <node id="14" value="membaca inputan"/>
            <node id="15" value="menghitung luas" />
            <node id="16" value="menghitung keliling/>
            <node id="17" value="menampilkan hasil"/>
            <node id="18" value="menampilkan jenis pilihan"/>
            <node id="19" value="menampilkan luas"/>
            <node id="20" value="menampilkan keliling" />
            <node id="21" value="pilihan=2" />
            <node id="22" value="deklarasi var jari2"/>
            <node id="23" value="jenisyang dipilih/>
            <node id="24" value="inputkan jari2"/>
            <node id="25" value="membaca inputan"/>
            <node id="26" value="menghitung luas" />
            <node id="27" value="menghitung keliling"/>
            <node id="28" value="menampilkan hasil"/>
            <node id="29" value="menampilkan jenis pilihan"/>
            <node id="30" value="menampilkan luas"/>
            <node id="31" value="menampilkan keliling" />
            <node id="32" value="menampilkan pilihan selain 1/2"/>
            <node id="33" value="membaca inputan string"/>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="3to4" source="3" target="4" />
            <edge id="4to5" source="4" target="5" />
            <edge id="5to6" source="5" target="6" />
            <edge id="6to7" source="6" target="7" />
            <edge id="7to8" source="7" target="8" />
            <edge id="8to9" source="8" target="9" />
            <edge id="9to10" source="9" target="10" />
            <edge id="10to21" source="10" target="21" />
            <edge id="10to11" source="10" target="11" />
            <edge id="11to12" source="11" target="12" />
            <edge id="12to13" source="12" target="13" />
            <edge id="13to14" source="13" target="14" />
            <edge id="14to15" source="14" target="15" />
            <edge id="15to16" source="15" target="16" />
            <edge id="16to17" source="16" target="17" />
            <edge id="17to18" source="17" target="18" />
            <edge id="18to19" source="18" target="19" />
            <edge id="19to20" source="19" target="20" />
            <edge id="20to33" source="20" target="33" />
            <edge id="21to32" source="21" target="32" />
            <edge id="21to22" source="21" target="22" />
            <edge id="22to23" source="22" target="23" />
            <edge id="23to24" source="23" target="24" />
            <edge id="24to25" source="24" target="25" />
            <edge id="25to26" source="25" target="26" />
            <edge id="26to27" source="26" target="27" />
            <edge id="27to28" source="27" target="28" />
            <edge id="28to29" source="28" target="29" />
            <edge id="29to30" source="29" target="30" />
            <edge id="30to31" source="30" target="31" />
            <edge id="31to33" source="31" target="33" />
            <edge id="32to33" source="32" target="33" />
      </edges>
  </graph>
</graphml>
```

6. GML program ATM

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed" parse.nodes="33"
parse.edges="38" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
    <nodes>
            <node id="0" value="deklarasi variabel integer" />
            <node id="1" value="deklarasi variabel integer"/>
            <node id="2" value="inputkan pin"/>
            <node id="3" value="membaca inputan"/>
            <node id="4" value="menmapilkan Welcome to ATM
            Service"/>
            <node id="5" value="menampilkan pilihan 1. Check
            Balance" />
            <node id="6" value="menampilkan pilihan 2. Withdraw
            Cash"/>
            <node id="7" value="menampilkan pilihan 3. Deposit
            Cash"/>
            <node id="8" value="menampilkan pilihan 4. Quit"/>
            <node id="9" value="menampilkan *** />
            <node id="10" value="inputkan pilihan" />
            <node id="11" value="membaca inputan"/>
            <node id="12" value="menampilkan inputan"/>
            <node id="13" value="pilihan 1"/>
            <node id="14" value="menampilkan saldo"/>
            <node id="15" value="pilihan 2" />
            <node id="16" value="inputkan jumlah penarikan uang/>
            <node id="17" value="membaca inputan"/>
            <node id="18" value="cek apakah inputan kelipatan
            100)"/>
            <node id="19" value="manampilkan nonimal harus
            kelipatan 100"/>
            <node id="20" value="jika nominal penarikan > (saldo -
            500)" />
            <node id="21" value="menampilkan saldo tidak mencukupi"
            />
            <node id="22" value="pengurangan saldo"/>
            <node id="23" value="menampilkan saran menambah
            tabungan"/>
            <node id="24" value="menampilkan saldo"/>
            <node id="25" value="pilihan 3"/>
            <node id="26" value="inputkan nominal tabungan" />
            <node id="27" value="membaca inputan"/>
            <node id="28" value="menambahkan saldo"/>
            <node id="29" value="menampilkan saldo"/>
            <node id="30" value="pilihan 4" />
            <node id="31" value="menampilkan ucapan terimakasih;"/>
            <node id="32" value="membaca inputan string"/>
      </nodes>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="3to4" source="3" target="4" />
            <edge id="4to5" source="4" target="5" />
            <edge id="5to6" source="5" target="6" />
            <edge id="6to7" source="6" target="7" />
            <edge id="7to8" source="7" target="8" />
            <edge id="8to9" source="8" target="9" />
```

```
            <edge id="9to10" source="9" target="10" />
            <edge id="10to11" source="10" target="11" />
            <edge id="11to12" source="11" target="12" />
            <edge id="12to13" source="12" target="13" />

            <edge id="13to15" source="13" target="15" />
            <edge id="13to14" source="13" target="14" />
            <edge id="14to32" source="14" target="32" />

            <edge id="15to25" source="15" target="25" />
            <edge id="15to16" source="15" target="16" />
            <edge id="16to17" source="16" target="17" />
            <edge id="17to18" source="17" target="18" />
            <edge id="18to20" source="18" target="20" />
            <edge id="18to19" source="18" target="19" />
            <edge id="19to32" source="19" target="32" />

            <edge id="20to22" source="20" target="22" />
            <edge id="20to21" source="20" target="21" />
            <edge id="21to32" source="21" target="32" />
            <edge id="22to23" source="22" target="23" />
            <edge id="23to24" source="23" target="24" />
            <edge id="24to32" source="24" target="32" />

            <edge id="25to30" source="25" target="30" />
            <edge id="25to26" source="25" target="26" />
            <edge id="26to27" source="26" target="27" />
            <edge id="27to28" source="27" target="28" />
            <edge id="28to29" source="28" target="29" />
            <edge id="29to32" source="29" target="32" />

            <edge id="30to31" source="30" target="31" />
            <edge id="30to32" source="30" target="32" />
            <edge id="31to32" source="31" target="32" />
      </edges>
  </graph>
</graphml>
```

7. GML program perulangan menampilkan bilangan dari 1 sampai 10

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed" parse.nodes="7"
parse.edges="8" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
    <nodes>
            <node id="0" value="deklarasi variabel int i" />
            <node id="1" value="menampilkan tulisan Bilangan =" />
            <node id="2" value="cek i<=2 "/>
            <node id="3" value="Simpan bilangan"/>
            <node id="4" value="menambahkan bilangan dengan
            variabel i"/>
            <node id="5" value="increment variabel i"/>
            <node id="6" value="menampilkan hasil" />
            <node id="7" value="membaca inputan string" />
      </nodes>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
```

```
            <edge id="2to5" source="2" target="5" />
            <edge id="3to4" source="3" target="4" />
            <edge id="4to2" source="4" target="2" />
            <edge id="4to5" source="4" target="5" />
            <edge id="5to6" source="5" target="6" />
      </edges>
  </graph>
</graphml>
```

8. GML program perulangan menampilkan bilangan genap dari 1 sampai 10

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed" parse.nodes="9"
parse.edges="11" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
    <nodes>
            <node id="0" value="deklarasi variabel integer i" />
            <node id="1" value="menampilkan bilangan genap" />
            <node id="2" value="perulangan cek i<=10"/>
            <node id="3" value="cek jika i%2==0"/>
            <node id="4" value="simpan nilai i"/>
            <node id="5" value="menambahkan bilangan dengan
            variabel i"/>
            <node id="6" value="menampilkan hasil" />
            <node id="7" value="Smenampilkan jumlah bilangan genap"
            />
            <node id="8" value="membaca inputan string" />
      </nodes>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="2to6" source="2" target="6" />
            <edge id="3to4" source="3" target="4" />
            <edge id="3to2" source="3" target="2" />
            <edge id="4to5" source="4" target="5" />
            <edge id="5to6" source="5" target="6" />
            <edge id="5to2" source="5" target="2" />
            <edge id="6to7" source="6" target="7" />
            <edge id="7to8" source="7" target="8" />
      </edges>
  </graph>
</graphml>
```

9. GML program menghitung jumlah bilangan yang diinputkan

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed" parse.nodes="11"
parse.edges="12" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
    <nodes>
<node id="0" value="deklarsi variabel i, n" />
            <node id="1" value="inputkan banyaknya bilangan"/>
            <node id="2" value="membaca inputan"/>
            <node id="3" value="perulangan i <= n)"/>
            <node id="4" value="menampilkan bilangan ke"/>
            <node id="5" value="membaca inputan" />
            <node id="6" value="menghitung penjumlahan"/>
            <node id="7" value="increment variabel i"/>
```

```
            <node id="8" value="menampilkan hasil"/>
            <node id="9" value="menampilkan hasil penjumlahan" />
            <node id="10" value="mambaca inputan string"/>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="3to4" source="3" target="4" />
            <edge id="3to8" source="3" target="8" />
            <edge id="4to5" source="4" target="5" />
            <edge id="5to6" source="5" target="6" />
            <edge id="6to7" source="6" target="7" />
            <edge id="7to3" source="7" target="3" />
            <edge id="7to8" source="7" target="8" />
            <edge id="8to9" source="8" target="9" />
            <edge id="9to10" source="9" target="10" />
      </edges>
  </graph>
</graphml>
```

10. GML program menghitung nilai rata- rata

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed" parse.nodes="19"
parse.edges="20" parse.order="nodesfirst" parse.nodeids="free"
parse.edgeids="free">
    <nodes>
            <node id="0" value="deklarasai variabel array (a)" />
            <node id="1" value="deklarasi variabel integer"/>
            <node id="2" value="value array [1]"/>
            <node id="3" value="value array [2]"/>
            <node id="4" value="value array [3]"/>
            <node id="5" value="value array [4]" />
            <node id="6" value="value array [5]"/>
            <node id="7" value="set nilai variabel integer"/>
            <node id="8" value="set nilai variabel rata2"/>
            <node id="9" value="set nilai variabel total" />
            <node id="10" value="menampilkan nilai tersimpan" />
            <node id="11" value="perulangan a<5)"/>
            <node id="12" value="menjumlahkan nilai"/>
            <node id="13" value="menghitung total"/>
            <node id="14" value="increment variabel a"/>
            <node id="15" value="menampilkan total nilai" />
            <node id="16" value="menghitung rata2"/>
            <node id="17" value="menampilkan rata2"/>
            <node id="18" value="membaca inputan string"/>
      </nodes>
      <edges>
            <edge id="0to1" source="0" target="1" />
            <edge id="1to2" source="1" target="2" />
            <edge id="2to3" source="2" target="3" />
            <edge id="3to4" source="3" target="4" />
            <edge id="4to5" source="4" target="5" />
            <edge id="5to6" source="5" target="6" />
            <edge id="6to7" source="6" target="7" />
            <edge id="7to8" source="7" target="8" />
            <edge id="8to9" source="8" target="9" />
            <edge id="9to10" source="9" target="10" />
            <edge id="10to11" source="10" target="11" />
```

```
            <edge id="11to12" source="11" target="12" />
            <edge id="11to15" source="11" target="15" />
            <edge id="12to13" source="12" target="13" />
            <edge id="13to14" source="13" target="14" />
            <edge id="14to11" source="14" target="11" />
            <edge id="14to15" source="14" target="15" />
            <edge id="15to16" source="15" target="16" />
            <edge id="16to17" source="16" target="17" />
            <edge id="17to18" source="17" target="18" />
        </edges>
    </graph>
</graphml>
```

LAMPIRAN 4 Kasus Uji

1. Kasus uji program menghitung penjumlahan, pengurangan, perkalian, dan pembagian dari 2 buah variabel yang diinputkan

Kasus uji = 1, yaitu 0->1->2->3->4->5->6->7->8->9->10->11->12->13->14

Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer"/>
<node id="1" value="deklarasi variabel desimal
<node id="2" value="inputan pertama"/>
<node id="3" value="membaca inputan"/>
<node id="4" value="inputan kedua"/>
<node id="5" value="membaca inputan"/>
<node id="6" value="rumus penjumlahan"/>
<node id="7" value="rumus pengurangan"/>
<node id="8" value="rumus perkalian"/>
<node id="9" value="rumus pembagian"/>
<node id="10" value="menampilkan hasil penjumlahan"/>
<node id="11" value="menampilkan hasil pengurangan"/>
<node id="12" value="menampilkan hasil perkalian"/>
<node id="13" value="menampilkan hasil pembagian"/>
<node id="14" value="membaca inputan string"/>
```

2. Kasus uji program menentukan bilangan genap dan ganjil sebuah bilangan

Kasus Uji = 2

0->1->2->3->4->6. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel"/>
<node id="1" value="input bilangan (int)"/>
<node id="2" value="membaca inputan integer"/>
<node id="3" value="cek bil % 2 == 0"/>
<node id="4" value="menampilkan status bilangan ganjil);"/>
<node id="6" value="membaca inputan string"/>
```

0->1->2->3->5->6. Urutan langkah kasus uji =

```
node id="0" value="deklarasi variabel"/>
<node id="1" value="input bilangan (int)"/>
<node id="2" value="membaca inputan integer"/>
<node id="3" value="cek bil % 2 == 0"/>
<node id="5" value="menampilkan status bilangan genap);"/>
<node id="6" value="membaca inputan string"/>
```

3. Kasus uji program menentukan predikat suatu nilai

Kasus uji = 5

0->1->2->3->4->12. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel nilai "/>
<node id="1" value="memasukkan nilai "/>
<node id="2" value="membca inputan nilai"/>
<node id="3" value="cek (nilai >=80)dan(nilai <= 100)"/>
```

```
<node id="4" value="menampilkan nilai A"/>
<node id="12" value="membaca inputan string"/>
```

0->1->2->3->5->6->12. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel nilai "/>
<node id="1" value="memasukkan nilai "/>
<node id="2" value="membca inputan nilai"/>
<node id="3" value="cek (nilai >=80)dan(nilai <= 100)"/>
<node id="5" value="jika (nilai >= 70) dan (nilai <= 80)"/>
<node id="6" value="menampilkan nilai B);"/>
<node id="12" value="membaca inputan string"/>
```

0->1->2->3->5->7->8->12. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel nilai "/>
<node id="1" value="memasukkan nilai "/>
<node id="2" value="membca inputan nilai"/>
<node id="3" value="cek (nilai >=80)dan(nilai <= 100)"/>
<node id="5" value="jika (nilai >= 70) dan (nilai <= 80)"/>
<node id="7" value="jika (nilai >= 60) dan (nilai <= 70)"/>
<node id="8" value="menampilkan nilai C"/>
<node id="12" value="membaca inputan string"/>
```

0->1->2->3->5->7->9->10->12. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel nilai "/>
<node id="1" value="memasukkan nilai "/>
<node id="2" value="membca inputan nilai"/>
<node id="3" value="cek (nilai >=80)dan(nilai <= 100)"/>
<node id="5" value="jika (nilai >= 70) dan (nilai <= 80)"/>
<node id="7" value="jika (nilai >= 60) dan (nilai <= 70)"/>
<node id="9" value="jika (nilai >= 50) dan (nilai <= 60))"/>
<node id="10" value="menampilkan nilai D"/>
<node id="12" value="membaca inputan string"/>
```

0->1->2->3->5->7->9->11->12. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel nilai "/>
<node id="1" value="memasukkan nilai "/>
<node id="2" value="membca inputan nilai"/>
<node id="3" value="cek (nilai >=80)dan(nilai <= 100)"/>
<node id="5" value="jika (nilai >= 70) dan (nilai <= 80)"/>
<node id="7" value="jika (nilai >= 60) dan (nilai <= 70)"/>
<node id="9" value="jika (nilai >= 50) dan (nilai <= 60))"/>
<node id="11" value="jika <50, menampilkan nilai E"/>
<node id="12" value="membaca inputan string"/>
```

4. Kasus uji program menentukan nilai terbesar dari 3 buah bilangan

Kasus uji = 3

0->1->2->3->4->5->6->7->8->9->13. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="inputan pertama (a)"/>
<node id="2" value="membaca  inputan pertama"/>
<node id="3" value="inputan kedua (b)"/>
<node id="4" value="membaca inputan kedua"/>
<node id="5" value="inputan ketiga"/>
```

```
<node id="6" value="membaca inputan ketiga (c)"/>
<node id="7" value="menampilkan garis batas"/>
<node id="8" value="cek a>c dan a>b)"/>
<node id="9" value="menampilkan inputan pertama(a)" />
<node id="13" value="membaca inputan string" />
```

0->1->2->3->4->5->6->7->8->10->11->13. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="inputan pertama (a)"/>
<node id="2" value="membaca  inputan pertama"/>
<node id="3" value="inputan kedua (b)"/>
<node id="4" value="membaca inputan kedua"/>
<node id="5" value="inputan ketiga"/>
<node id="6" value="membaca inputan ketiga (c)"/>
<node id="7" value="menampilkan garis batas"/>
<node id="8" value="cek a>c dan a>b)"/>
<node id="10" value="cek jika b>a dan b>c" />
<node id="11" value="menampilkan inputan kedua (b)"/>
<node id="13" value="membaca inputan string" />
```

0->1->2->3->4->5->6->7->8->10->12->13. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="inputan pertama (a)"/>
<node id="2" value="membaca  inputan pertama"/>
<node id="3" value="inputan kedua (b)"/>
<node id="4" value="membaca inputan kedua"/>
<node id="5" value="inputan ketiga"/>
<node id="6" value="membaca inputan ketiga (c)"/>
<node id="7" value="menampilkan garis batas"/>
<node id="8" value="cek a>c dan a>b)"/>
<node id="10" value="cek jika b>a dan b>c" />
<node id="12" value="menampilkan inputan ketiga (c)"/>
<node id="13" value="membaca inputan string" />
```

5. Kasus uji program menghitung luas persegi dan lingkaran

Kasus uji = 3

0->1->2->3->4->5->6->7->8->9->10->21->32->33. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel double"/>
<node id="2" value="deklarasi variabel string"/>
<node id="3" value="deklarasi variabel phi"/>
<node id="4" value="menampilkan judul aplikasi"/>
<node id="5" value="pilihan 1 = bujur sangkar);" />
<node id="6" value="pilihan 2 = lingkaran);"/>
<node id="7" value="inputkan pilihan"/>
<node id="8" value="membaca inputan"/>
<node id="9" value="menampilkan inputan" />
<node id="10" value="pilihan=1" />
<node id="21" value="pilihan=2" />
<node id="32" value="menampilkan pilihan selain 1/2"/>
<node id="33" value="membaca inputan string"/>
```

0->1->2->3->4->5->6->7->8->9->10->21->22->23->24->25->26->27->28->29 ->30->31->33. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel double"/>
<node id="2" value="deklarasi variabel string"/>
<node id="3" value="deklarasi variabel phi"/>
<node id="4" value="menampilkan judul aplikasi"/>
<node id="5" value="pilihan 1 = bujur sangkar);" />
<node id="6" value="pilihan 2 = lingkaran);"/>
<node id="7" value="inputkan pilihan"/>
<node id="8" value="membaca inputan"/>
<node id="9" value="menampilkan inputan" />
<node id="10" value="pilihan=1" />
<node id="21" value="pilihan=2" />
<node id="22" value="deklarasi var jari2"/>
<node id="23" value="jenisyang dipilih/>
<node id="24" value="inputkan jari2"/>
<node id="25" value="membaca inputan"/>
<node id="26" value="menghitung luas" />
<node id="27" value="menghitung keliling"/>
<node id="28" value="menampilkan hasil"/>
<node id="29" value="menampilkan jenis pilihan"/>
<node id="30" value="menampilkan luas"/>
<node id="31" value="menampilkan keliling" />
<node id="33" value="membaca inputan string"/>
```

0->1->2->3->4->5->6->7->8->9->10->11->12->13->14->15->16->17->18 ->19->20->33. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel double"/>
<node id="2" value="deklarasi variabel string"/>
<node id="3" value="deklarasi variabel phi"/>
<node id="4" value="menampilkan judul aplikasi"/>
<node id="5" value="pilihan 1 = bujur sangkar);" />
<node id="6" value="pilihan 2 = lingkaran);"/>
<node id="7" value="inputkan pilihan"/>
<node id="8" value="membaca inputan"/>
<node id="9" value="menampilkan inputan" />
<node id="10" value="pilihan=1" />
<node id="11" value="deklarasi var sisi"/>
<node id="12" value="jenis yang dipilih"/>
<node id="13" value="inputkan sisi"/>
<node id="14" value="membaca inputan"/>
<node id="15" value="menghitung luas" />
<node id="16" value="menghitung keliling/>
<node id="17" value="menampilkan hasil"/>
<node id="18" value="menampilkan jenis pilihan"/>
<node id="19" value="menampilkan luas"/>
<node id="20" value="menampilkan keliling" />
<node id="33" value="membaca inputan string"/>
```

6. Kasus uji program ATM

Kasus uji =7

0->1->2->3->4->5->6->7->8->9->10->11->12->13->15->25->30->31->32. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel integer"/>
<node id="2" value="inputkan pin"/>
<node id="3" value="membaca inputan"/>
```

```
<node id="4" value="menmapilkan Welcome to ATM Service"/>
<node id="5" value="menampilkan pilihan 1. Check Balance" />
<node id="6" value="menampilkan pilihan 2. Withdraw Cash"/>
<node id="7" value="menampilkan pilihan 3. Deposit Cash"/>
<node id="8" value="menampilkan pilihan 4. Quit"/>
<node id="9" value="menampilkan *** />
<node id="10" value="inputkan pilihan" />
<node id="11" value="membaca inputan"/>
<node id="12" value="menampilkan inputan"/>
<node id="13" value="pilihan 1"/>
<node id="15" value="pilihan 2" />
<node id="25" value="pilihan 3"/>
<node id="30" value="pilihan 4" />
<node id="31" value="menampilkan ucapan terimakasih;"/>
<node id="32" value="membaca inputan string"/>
```

0->1->2->3->4->5->6->7->8->9->10->11->12->13->15->25->30->32. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel integer"/>
<node id="2" value="inputkan pin"/>
<node id="3" value="membaca inputan"/>
<node id="4" value="menmapilkan Welcome to ATM Service"/>
<node id="5" value="menampilkan pilihan 1. Check Balance" />
<node id="6" value="menampilkan pilihan 2. Withdraw Cash"/>
<node id="7" value="menampilkan pilihan 3. Deposit Cash"/>
<node id="8" value="menampilkan pilihan 4. Quit"/>
<node id="9" value="menampilkan *** />
<node id="10" value="inputkan pilihan" />
<node id="11" value="membaca inputan"/>
<node id="12" value="menampilkan inputan"/>
<node id="13" value="pilihan 1"/>
<node id="15" value="pilihan 2" />
<node id="25" value="pilihan 3"/>
<node id="30" value="pilihan 4" />
<node id="32" value="membaca inputan string
```

0->1->2->3->4->5->6->7->8->9->10->11->12->13->15->25->26->27->28->29 ->32. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel integer"/>
<node id="2" value="inputkan pin"/>
<node id="3" value="membaca inputan"/>
<node id="4" value="menmapilkan Welcome to ATM Service"/>
<node id="5" value="menampilkan pilihan 1. Check Balance" />
<node id="6" value="menampilkan pilihan 2. Withdraw Cash"/>
<node id="7" value="menampilkan pilihan 3. Deposit Cash"/>
<node id="8" value="menampilkan pilihan 4. Quit"/>
<node id="9" value="menampilkan *** />
<node id="10" value="inputkan pilihan" />
<node id="11" value="membaca inputan"/>
<node id="12" value="menampilkan inputan"/>
<node id="13" value="pilihan 1"/>
<node id="15" value="pilihan 2" />
<node id="25" value="pilihan 3"/>
<node id="26" value="inputkan nominal tabungan" />
<node id="27" value="membaca inputan"/>
<node id="28" value="menambahkan saldo"/>
```

```
<node id="29" value="menampilkan saldo"/>
<node id="32" value="membaca inputan string
```

0->1->2->3->4->5->6->7->8->9->10->11->12->13->15->16->17->18->20->22->23->24->32. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel integer"/>
<node id="2" value="inputkan pin"/>
<node id="3" value="membaca inputan"/>
<node id="4" value="menmapilkan Welcome to ATM Service"/>
<node id="5" value="menampilkan pilihan 1. Check Balance" />
<node id="6" value="menampilkan pilihan 2. Withdraw Cash"/>
<node id="7" value="menampilkan pilihan 3. Deposit Cash"/>
<node id="8" value="menampilkan pilihan 4. Quit"/>
<node id="9" value="menampilkan *** />
<node id="10" value="inputkan pilihan" />
<node id="11" value="membaca inputan"/>
<node id="12" value="menampilkan inputan"/>
<node id="13" value="pilihan 1"/>
<node id="15" value="pilihan 2" />
<node id="16" value="inputkan jumlah penarikan uang/>
<node id="17" value="membaca inputan"/>
<node id="18" value="cek apakah inputan kelipatan 100)"/>
<node id="20" value="jika nominal penarikan > (saldo - 500)" />
<node id="22" value="pengurangan saldo"/>
<node id="23" value="menampilkan saran menambah tabungan"/>
<node id="24" value="menampilkan saldo"/>
<node id="32" value="membaca inputan string
```

0->1->2->3->4->5->6->7->8->9->10->11->12->13->15->16->17->18->20->21->32. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel integer"/>
<node id="2" value="inputkan pin"/>
<node id="3" value="membaca inputan"/>
<node id="4" value="menmapilkan Welcome to ATM Service"/>
<node id="5" value="menampilkan pilihan 1. Check Balance" />
<node id="6" value="menampilkan pilihan 2. Withdraw Cash"/>
<node id="7" value="menampilkan pilihan 3. Deposit Cash"/>
<node id="8" value="menampilkan pilihan 4. Quit"/>
<node id="9" value="menampilkan *** />
<node id="10" value="inputkan pilihan" />
<node id="11" value="membaca inputan"/>
<node id="12" value="menampilkan inputan"/>
<node id="13" value="pilihan 1"/>
<node id="15" value="pilihan 2" />
<node id="16" value="inputkan jumlah penarikan uang/>
<node id="17" value="membaca inputan"/>
<node id="18" value="cek apakah inputan kelipatan 100)"/>
<node id="20" value="jika nominal penarikan > (saldo - 500)" />
<node id="21" value="menampilkan saldo tidak mencukupi" />
<node id="32" value="membaca inputan string”/>
```

0->1->2->3->4->5->6->7->8->9->10->11->12->13->15->16->17->18->19->32. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel integer"/>
```

```
<node id="2" value="inputkan pin"/>
<node id="3" value="membaca inputan"/>
<node id="4" value="menmapilkan Welcome to ATM Service"/>
<node id="5" value="menampilkan pilihan 1. Check Balance" />
<node id="6" value="menampilkan pilihan 2. Withdraw Cash"/>
<node id="7" value="menampilkan pilihan 3. Deposit Cash"/>
<node id="8" value="menampilkan pilihan 4. Quit"/>
<node id="9" value="menampilkan *** />
<node id="10" value="inputkan pilihan" />
<node id="11" value="membaca inputan"/>
<node id="12" value="menampilkan inputan"/>
<node id="13" value="pilihan 1"/>
<node id="15" value="pilihan 2" />
<node id="16" value="inputkan jumlah penarikan uang/>
<node id="17" value="membaca inputan"/>
<node id="18" value="cek apakah inputan kelipatan 100)"/>
<node id="19" value="manampilkan nonimal harus kelipatan 100"/>
<node id="32" value="membaca inputan string”/>
```

0->1->2->3->4->5->6->7->8->9->10->11->12->13->14->32. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer" />
<node id="1" value="deklarasi variabel integer"/>
<node id="2" value="inputkan pin"/>
<node id="3" value="membaca inputan"/>
<node id="4" value="menmapilkan Welcome to ATM Service"/>
<node id="5" value="menampilkan pilihan 1. Check Balance" />
<node id="6" value="menampilkan pilihan 2. Withdraw Cash"/>
<node id="7" value="menampilkan pilihan 3. Deposit Cash"/>
<node id="8" value="menampilkan pilihan 4. Quit"/>
<node id="9" value="menampilkan *** />
<node id="10" value="inputkan pilihan" />
<node id="11" value="membaca inputan"/>
<node id="12" value="menampilkan inputan"/>
<node id="13" value="pilihan 1"/>
<node id="14" value="menampilkan saldo"/>
<node id="32" value="membaca inputan string”/>
```

7. Kasus uji program perulangan menampilkan bilangan dari 1 sampai 10

Kasus uji = 2

0->1->2->3->4->5->6->7. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel int i" />
<node id="1" value="menampilkan tulisan Bilangan =" />
<node id="2" value="cek i<=2 "/>
<node id="3" value="Simpan bilangan"/>
<node id="4" value="menambahkan bilangan dengan variabel i"/>
<node id="5" value="increment variabel i"/>
<node id="6" value="menampilkan hasil" />
<node id="7" value="membaca inputan string" />
```

0->1->2->6->7. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel int i" />
<node id="1" value="menampilkan tulisan Bilangan =" />
<node id="2" value="cek i<=2 "/>
<node id="6" value="menampilkan hasil" />
<node id="7" value="membaca inputan string" />
```

8. Kasus uji program perulangan menampilkan bilangan genap dari 1 sampai 10

Kasus uji = 2

0->1->2->3->4->5->6->7->8. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer i" />
<node id="1" value="menampilkan bilangan genap" />
<node id="2" value="perulangan cek i<=10"/>
<node id="3" value="cek jika i%2==0"/>
<node id="4" value="simpan nilai i"/>
<node id="5" value="menambahkan bilangan dengan variabel i"/>
<node id="6" value="menampilkan hasil" />
<node id="7" value="Smenampilkan jumlah bilangan genap" />
<node id="8" value="membaca inputan string" />
```

0->1->2->6->7->8. Urutan langkah kasus uji =

```
<node id="0" value="deklarasi variabel integer i" />
<node id="1" value="menampilkan bilangan genap" />
<node id="2" value="perulangan cek i<=10"/>
<node id="6" value="menampilkan hasil" />
<node id="7" value="Smenampilkan jumlah bilangan genap" />
<node id="8" value="membaca inputan string" />
```

9. Kasus uji program menghitung jumlah bilangan yang diinputkan

Kasus uji = 2

0->1->2->3->4->5->6->7->8->9->10. Urutan langkah kasus uji =

```
<node id="1" value="inputkan banyaknya bilangan"/>
<node id="2" value="membaca inputan"/>
<node id="3" value="perulangan i <= n)"/>
<node id="4" value="menampilkan bilangan ke"/>
<node id="5" value="membaca inputan" />
<node id="6" value="menghitung penjumlahan"/>
<node id="7" value="increment variabel i"/>
<node id="8" value="menampilkan hasil"/>
<node id="9" value="menampilkan hasil penjumlahan" />
<node id="10" value="mambaca inputan string"/>
```

0->1->2->3->8->9->10. Urutan langkah kasus uji =

```
<node id="1" value="inputkan banyaknya bilangan"/>
<node id="2" value="membaca inputan"/>
<node id="3" value="perulangan i <= n)"/>
<node id="8" value="menampilkan hasil"/>
<node id="9" value="menampilkan hasil penjumlahan" />
<node id="10" value="mambaca inputan string"/>
```

10. Kasus uji program menghitung nilai rata- rata

Kasus uji = 2

0->1->2->3->4->5->6->7->8->9->10->11->12->13->14->15->16->17->18. Urutan langkah kasus uji =

```
<node id="0" value="deklarasai variabel array (a)" />
<node id="1" value="deklarasi variabel integer"/>
<node id="2" value="value array [1]"/>
<node id="3" value="value array [2]"/>
<node id="4" value="value array [3]"/>
<node id="5" value="value array [4]" />
<node id="6" value="value array [5]"/>
<node id="7" value="set nilai variabel integer"/>
<node id="8" value="set nilai variabel rata2"/>
<node id="9" value="set nilai variabel total" />
<node id="10" value="menampilkan nilai tersimpan" />
<node id="11" value="perulangan a<5)"/>
<node id="12" value="menjumlahkan nilai"/>
<node id="13" value="menghitung total"/>
<node id="14" value="increment variabel a"/>
<node id="15" value="menampilkan total nilai" />
<node id="16" value="menghitung rata2"/>
<node id="17" value="menampilkan rata2"/>
<node id="18" value="membaca inputan string"/>
```

0->1->2->3->4->5->6->7->8->9->10->11->15->16->17->18. Urutan langkah kasus uji =

```
<node id="0" value="deklarasai variabel array (a)" />
<node id="1" value="deklarasi variabel integer"/>
<node id="2" value="value array [1]"/>
<node id="3" value="value array [2]"/>
<node id="4" value="value array [3]"/>
<node id="5" value="value array [4]" />
<node id="6" value="value array [5]"/>
<node id="7" value="set nilai variabel integer"/>
<node id="8" value="set nilai variabel rata2"/>
<node id="9" value="set nilai variabel total" />
<node id="10" value="menampilkan nilai tersimpan" />
<node id="11" value="perulangan a<5)"/>
<node id="15" value="menampilkan total nilai" />
<node id="16" value="menghitung rata2"/>
<node id="17" value="menampilkan rata2"/>
<node id="18" value="membaca inputan string"/>
```

LAMPIRAN 5 Data Uji

1. Data uji program menghitung penjumlahan, pengurangan, perkalian, dan pembagian dari 2 buah variabel yang diinputkan

No		Masukan	Keluaran	Kasus Uji
1	VALID	10, 2	12, 8, 20, 5	Konsisten
2	VALID	20, -5	15, 25, -100, -4	Konsisten

2. Data uji program menentukan bilangan genap dan ganjil sebuah bilangan

No		Masukan	Keluaran	Kasus Uji
1	VALID	0	Bilangan genap	Konsisten
2		-5	Bilangan ganjil	Konsisten
3		-100	Bilangan genap	Konsisten
4		2147483647	Bilangan ganjil	Konsisten
5		-2147483648	Bilangan genap	Konsisten

3. Data uji program menentukan predikat suatu nilai

No		Masukan	Keluaran	Kasus Uji
1	VALID	90	A	Konsisten
2	VALID	75	B	Konsisten
3	VALID	65	C	Konsisten
4	VALID	55	D	Konsisten
5	VALID	40	E	Konsisten

4. Data uji program menentukan nilai terbesar dari 3 buah bilangan

No		Masukan	Keluaran	Kasus Uji
1	VALID	30,23,25	A	Konsisten
2	VALID	50,100,25	B	Konsisten
3	VALID	45, 50, 90	C	Konsisten
4	VALID	-10,-15,-20	A	Konsisten

5. Data uji program menghitung luas persegi dan lingkaran

No		Masukan	Keluaran	Kasus Uji
1	VALID	3	Selain 1 atau 2	Konsisten
2	VALID	2	Lingkaran	Konsisten
3	VALID	1	Bujut sangkar	Konsisten

6. Data uji program ATM

No		Masukan	Keluaran	Kasus Uji
1	VALID	5	null	Konsisten
2	VALID	4	keluar sstem	Konsisten
3	VALID	3, 1000	2000	Konsisten
4	VALID	2, 200	800	Konsisten
5	VALID	2, 800	insufficent balance	Konsisten
6	VALID	2, 50	Kelipatan 100	Konsisten
7	VALID	1, default	1000	Konsisten

7. Data uji program perulangan menampilkan bilangan mulai dari 1

No		Masukan	Keluaran	Kasus Uji
1	VALID	2	1,2	Tidak Konsisten
2	VALID	8	1,2,3,4,5,6,7,8	Tidak Konsisten
3	TIDAK VALID	0	null	Konsisten
4	TIDAK VALID	-2	null	Konsisten

8. Data uji program perulangan menampilkan bilangan genap dari 1 -10

No		Masukan	Keluaran	Kasus Uji
1	VALID	10	30	Tidak Konsisten
2	VALID	20	110	Tidak Konsisten
3	TIDAK VALID	0	null	Konsisten
4	TIDAK VALID	-10	null	Konsisten

9. Data uji program menghitung jumlah bilangan yang diinputkan

No		Masukan	Keluaran	Kasus Uji
1	VALID	3=10,12, 3	25	Tidak Konsisten
2	VALID	5=5,3,4,10,6	28	Tidak Konsisten
3	TIDAK VALID	0	0	Konsisten
4	TIDAK VALID	-2	0	Konsisten

10. Data uji program menghitung nilai rata- rata

No		Masukan	Keluaran	Kasus Uji
1	VALID	80,90,70,80,90	410, 82	Tidak Konsisten
2	TIDAK VALID	0	0	Konsisten

BIOGRAFI PENULIS

Penulis, Desy Candra Novitasari, lahir di kota Nganjuk pada tanggal 21 November 1992. Penulis adalah anak pertama dari dua bersaudara dan dibesarkan di kota Nganjuk, Jawa Timur.

Penulis menempuh pendidikan formal di SD Negeri Ganung Kidul 1 Nganjuk (1998-2004) SMPN 2 Nganjuk (2004-2007), dan SMA Negeri 1 Nganjuk (2007-2010). Pada tahun 2010-2014, penulis melanjutkan pendidikan S1 di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya, Jawa Timur.

Pada tahun 2015-2017, penulis melanjutkan pendidikan Magister S2 di jurusan yang sama, yaitu Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya, Jawa Timur.

Di Jurusan Teknik Informatika, penulis mengambil bidang minat Rekayasa Perangkat Lunak. Penulis juga aktif dalam organisasi kemahasiswaan seperti: Himpunan Mahasiswa Teknik Computer (HMTC). Penulis dapat dihubungi melalui alamat email desycandra.045@gmail.com