TESIS

PENGECEKAN KEMIRIPAN MODUL SOURCECODE MENGGUNAKAN ALGORITMA RUNNING KARP-RABIN GREEDY STRING TILING

SIMILARITY CHECKING OF SOURCE CODE MODULE USING RUNNING KARP-RABIN GREEDY STRING TILING ALGORITHM

RUDI SETIAWAN
10/308841/PPA/03365

PROGRAM STUDI S2 ILMU KOMPUTER
JURUSAN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUANALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA

2013

TESIS

PENGECEKAN KEMIRIPAN MODUL SOURCE CODE MENGGUNAKAN ALGORITMA RUNNING KARP-RABIN GREEDY STRING TILING

SIMILARITY CHECKING OF SOURCE CODE MODULE USING RUNNING KARP-RABIN GREEDY STRING TILING ALGORITHM

Diajukan untuk memenuhi salah satu syarat memperoleh derajat
Master of Computer Science

RUDI SETIAWAN
10/308841/PPA/03365

PROGRAM STUDI S2 ILMU KOMPUTER
JURUSAN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA

2013

HALAMAN PENGESAHAN

TESIS

PENGECEKAN KEMIRIPAN MODUL SOURCE CODE MENGGUNAKAN ALGORITMA RUNNING KARP-RABIN GREEDY STRING TILING

dipersiapkan dan disusun oleh

RUDI SETIAWAN
10/308841/PPA/03365

telah dipertahankan di depan Dewan Penguji

pada tanggal 10 Juni 2013

Susunan Dewan Penguji

Pembimbing Utama

Drs. Edi Winarko, M.Sc., Ph.D
NIP. 19630223198701002

Anggota Dewan Penguji Lain

Drs. Agus Harjoko, M.Sc., Ph.D
NIP. 196008041987031003

Dra. Diah Junia Eksi Palupi, M.S
NIP. 195506161981012001

Drs. Retantyo Wardoyo, M.Sc., Ph.D
NIP. 195903111983031005

Tesis ini telah diterima sebagai salah satu persyaratan
untuk memperoleh gelar *Master of Computer Science*
Tanggal 10 Juni 2013

Dra. Sri Hartati, M.Sc, Ph.D
Pengelola Program Studi S2 Ilmu Komputer

DAFTAR ISI

Daftar Gambar

Daftar Tabel

INTISARI

Pengecekan Kemiripan Modul Source Code Menggunakan Algoritma Running Karp-Rabin Greedy String Tiling

Oleh

Rudi Setiawan

10/308841/PPA/03365

Proses pengecekan kemiripan modul *source code*, jika dilakukan secara manual membutuhkan ketelitian dan waktu yang tidak singkat untuk dapat menyelesaikannya. Berdasarkan masalah tersebut, pada penelitian ini dirancang sebuah perangkat lunak dengan pendekatan *structure-based* menggunakan teknik *string matching* dengan algoritma Running Karp-Rabin Greedy String Tiling (RKR-GST) untuk melakukan pengecekan kemiripan dan menggunakan metode Dice Coefficient untuk mengukur tingkat kemiripan dari 2 modul *source code* hasil pembandingan.

Hasil pengujian dari sistem yang telah dirancang, didapatkan beberapa kesimpulan diantaranya, algoritma RKRGST yang diterapkan pada sistem ini mampu mengenali perubahan urutan *statement* maupun perubahan *statement* yang telah dilakukan, serta mampu pula mengenali *syntax* procedure uji yang diambil sebagian dari modul pembandingnya. Modifikasi dengan menambah komentar pada modul *source code*, serta modifikasi perubahan nama *procedure* yang dipanggil didalam *body* suatu *procedure*, juga mampu dikenali oleh sistem. Waktu proses yang dibutuhkan oleh sistem untuk menghasilkan output, bergantung pada banyaknya baris kode program yang terdapat pada modul *source code*.

Kata Kunci : Pengecekan modul source code, RKRGST, similarity source code

ABSTRACT

Similarity Checking of Source Code Module Using Running Karp-Rabin Greedy String Tiling Algorithm

By

Rudi Setiawan

10/308841/PPA/03365

Similarity checking process of source code module if done manually needs accuracy and needs long time to finish it. Based on that problem, in this research was designed a software with structure-based approach using string matching technique with Running Karp-Rabin Greedy String Tiling (RKR-GST) Algorithm to check the similarity and using Dice Coefficient method to measure the level of similarity from 2 results source code modules.

The result of testing from system that has been modified, obtained some conclusions which include RKRGST algorithm which applied on this system proven capable of recognizing the statement permutation or the change of system which has been done, and be able to recognize the syntax procedure testing that taken in part from its comparison module. Modification by adding the comment on source code module and the change of procedure name modification which accepted in body of procedure are be able to recognized by system. Time process which needed to produce output depends on the number of program code row that contained in source code module.

Keywords: Similarity Checking of Source code module, RKRGST


UNIVERSITAS
JOGJAKARTA

BAB I
PENDAHULUAN

1.1 Latar Belakang

Proses pengecekan kemiripan modul *source code* bertujuan pendeteksian bagian-bagian *source code* yang memiliki kesamaan antara kedua modul *source code* yang diuji. Dengan adanya pengecekan kemiripan ini, diharapkan tindakan plagiarisme *source code* dapat dikurangi. Kasus plagiarisme sering terjadi dikalangan mahasiswa dibeberapa universitas (Wagner, 2000). Jika telah terjadi kasus seperti ini, khususnya pada tugas pemrograman, maka proses pengkoreksian tugaspun akan menjadi lebih sulit apabila dilakukan secara manual. Butuh sebuah ketelitian dan waktu yang tidak singkat untuk mengecek semua hasil pekerjaan tugas pemrograman yang telah dikumpulkan dari para mahasiswa.

Untuk meningkatkan ketelitian dan efisiensi penggunaan waktu dalam proses pengecekan kemiripan modul *source code*, bantuan komputer sangat diperlukan. Dengan menggunakan bantuan komputer, hanya dibutuhkan inputan modul *source code* yang akan diperiksa ke dalam perangkat lunak, selanjutnya perangkat lunak akan secara otomatis mendeteksi kemiripan dari dua modul *source code* yang diuji kemiripannya.

Terdapat 2 (dua) teknik yang dapat diterapkan untuk melakukan pengecekan kemiripan *modul source code* yaitu *attribute-counting* dan *structure-based*. Pada metode *attribute-counting*, yang dibandingkan adalah ukuran kuantitatif beberapa matriks program, sedangkan pada metode *structure-based* yang dibandingkan adalah representasi dari struktur program, misalnya representasi linier berupa *string*, *parse tree*, *data flow*, dan lain-lain.

Penelitian telah membuktikan bahwa metode *attribute-counting* tidak bisa menggambarkan struktur program secara baik untuk membedakan *source code* hasil plagiarisme dan *source code* yang bukan merupakan hasil plagiarisme (Faidhi dan Robinson, 1987). Berdasarkan hal inilah yang mendorong dikembangkannya metode berbasis struktur.

Berdasarkan pada permasalahan yang ada, maka pada penelitian ini akan dirancang sebuah perangkat lunak dengan pendekatan *structure-based*, menggunakan teknik *string matching* dengan algoritma Running Karp-Rabin Greedy String Tiling (RKR-GST) untuk melakukan pengecekan kemiripan dan menggunakan metode Dice Coefficient untuk mengukur tingkat kemiripan dari 2 modul *source code* hasil pembandingan.

Algoritma Running Karp-Rabin Greedy String Tiling dipilih karena keunggulannya dalam menemukan bagian-bagian *substring* yang identik pada dua buah string tanpa terpengaruh urutan maupun posisi *substring*. Pada proses pencariannya algoritma ini menggunakan *hash value* yang dibentuk menggunakan fungsi *hash*.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas dapat diuraikan beberapa rumusan masalah sebagai berikut :

1. Bagaimana membangun suatu sistem yang dapat melakukan pengecekan tingkat kemiripan terhadap 2 (dua) modul *source code* dari suatu bahasa pemrograman.
2. Bagaimana mengimplementasikan metode pengecekan kemiripan antara 2 (dua) modul *source code* menggunakan algoritma Running Karp-Rabin Greedy String Tiling (RKRGST).

1.3 Batasan Masalah

Batasan masalah dalam penelitian ini adalah :

1. Data pengujian yang digunakan berupa data uji dari modul *source code* pada bahasa pemrograman VB.6 yang berupa file berekstensi .frm, VB.Net yang berekstensi .vb, dan C# berekstensi .cs, serta Borland Delphi berekstensi .pas.
2. Pengujian terhadap *source code* yang berbeda bahasa pemrograman tidak dapat dilakukan oleh sistem. Sistem hanya melakukan pengujian terhadap *source code* dari bahasa pemrograman yang sama.
3. Sistem tidak memperhatikan kesalahan *syntax* pada penulisan program.

4. Pernyataan kondisional seperti *if .. then .. else* dan *select .. case* dianggap sebagai *syntax* yang berbeda meski pada program memiliki kesamaan maksud dan tujuan.
5. Sistem tidak mampu mengenali pemanggilan *procedure/function/method* yang didefinisikan diluar modul yang sedang dibandingkan tanpa menggunakan *keyword "Call"*.
6. Sistem juga tidak mampu mengenali *filtering* variable yang bersifat global yang didefinisikan diluar modul *source code* yang sedang dibandingkan.

1.4 Tujuan dan Manfaat Penelitian

Tujuan dari penelitian ini adalah bagaimana menerapkan algoritma Running Karp-Rabin Greedy String Tiling (RKR-GST) untuk pengecekan kemiripan modul *source code* dan menghitung tingkat kemiripan kedua modul menggunakan metode *dice coefficient*.

Berdasarkan tujuan dari penelitian, diharapkan hasil penelitian ini dapat membantu petugas dalam mengoreksi kumpulan *source code* dari tugas pemrograman yang diberikan, serta dapat digunakan sebagai bahan pertimbangan dalam menentukan plagiarisme *source code* tetapi bukan sebagai acuan pengambilan keputusan terhadap adanya indikasi tindak plagiarisme *source code*.

1.5 Keaslian Penelitian

Penelitian mengenai plagiarisme *source code* telah banyak dilakukan bahkan dengan algoritma yang sama yaitu *Running Karp-Rabin Greedy String Tiling* seperti yang telah dilakukan oleh Kustanto (2009), dalam penelitiannya sebuah token mewakili satu baris *syntax* yang ada pada program, sedangkan yang dilakukan oleh peneliti saat ini proses pembentukan token berasal dari *syntax* yang berada di setiap *procedure* atau *function* yang ada pada modul *source code*, dengan terlebih dahulu dilakukan *filtering* terhadap keberadaan variable, *filtering* terhadap pemanggilan *procedure* yang ada didalam *body* dari suatu *procedure*, *filtering stopword*, *filtering* tanda baca, dan proses *case folding*.

Setelah melalui beberapa tahapan tersebut, kemudian akan dilakukan proses pembandingan terhadap setiap *syntax* yang ada pada *procedure* pembandingnya menggunakan algoritma Running Karp-Rabin Greedy String Tiling.

1.6 Metode Penelitian

Tahapan-tahapan dalam penelitian ini meliputi :

1.6.1 Pengumpulan data

Data yang digunakan sebagai pengujian adalah *source code* yang telah dilakukan beberapa modifikasi-modifikasi, diantaranya mengubah nama *indentifier*, merubah nama *procedure* atau *function*, merubah *type* data variable dengan *type* data lain yang sejenis, mengubah atau menambah komentar, mengubah pemanggilan suatu procedure dengan mengganti nama *procedure* yang dipanggil, merubah urutan *procedure* atau *function*, menyalin sebagian *source code* atau hanya sebagian *procedure* atau *function*, menyalin sebagian isi suatu *procedure* atau *function*.

1.6.2 Pengembangan sistem

Pada tahap pengembangan sistem dilakukan analisa kebutuhan-kebutuhan perangkat lunak dari sistem yang akan dibangun, sistem pengecekan kemiripan *modul source code* ini dirancang menggunakan teknik string matching dengan menerapkan algoritma Running Karp-Rabin Greedy String Tiling pada proses pencarian token-token yang memiliki kemiripan dimana token-token tersebut mewakili syntax dari setiap *procedure/function* yang ada pada masing-masing *source code* dan menggunakan metode dice coefficient untuk menghitung persentase kemiripan dari kedua *modul source code*, gambaran lebih rinci tentang pengembangan sistem akan dibahas pada bab IV.

1.6.3 Pengujian sistem

Proses pengujian sistem dilakukan untuk melihat kemampuan sistem dalam mengenali modifikasi-modifikasi yang telah dilakukan, dengan cara mengamati kesesuaian output sistem yang dihasilkan terhadap data pengujian.

1.6.4 Penulisan laporan

Setiap tahapan-tahapan yang dilakukan dalam penelitian dan pengembangan sistem selanjutnya akan disusun sebagai laporan tesis. Adapun tahapan-tahapan tersebut meliputi :

1. Mempelajari konsep algoritma Running Karp-Rabin Greedy String Tiling (RKRGST) dan mempelajari metode-metode penunjang lainnya yang akan digunakan untuk kepentingan dalam penelitian ini.
2. Menganalisa dan merancang sistem menggunakan metode-metode yang telah dipelajari sebelumnya.
3. Melakukan implementasi sistem berdasarkan analisa dan perancangan yang telah disusun.
4. Melakukan uji coba terhadap sistem yang telah dibuat dengan menganalisa hasil keluran sistem. Hasil yang dikeluarkan oleh sistem berupa persentase kemiripan serta waktu proses yang dibutuhkan oleh sistem dalam melakukan pembandingan.
5. Melaporkan pembahasan terhadap hasil uji coba yang telah dilakukan.
6. Menarik kesimpulan dari hasil uji coba sistem.

BAB II
TINJAUAN PUSTAKA

Beberapa aplikasi dan metode telah diciptakan dalam penelitian untuk kasus plagiarisme khususnya untuk plagiarisme *source code* antara lain :

Whale (1990) pada penelitiannya mengembangkan sistem yang diberinama Plague merupakan pengembangan dari pendekatan berbasis struktur dan menggunakan detail struktur program untuk proses pembandingan *source code*, hasil deteksi plagiarisme yang dihasilkan Plague cukup akurat, namun ada beberapa kekurangan yang dimiliki sebagai berikut :

1. Untuk menulis versi Plague baru agar dapat mengenali bahasa pemrograman lain, memerlukan usaha besar dan membutuhkan waktu yang tidak singkat, diawali dengan pembangunan parser untuk bahasa tujuan dan pemilihan *distance metrics* untuk penggunaan pada fase kedua.
2. Hasil proses deteksi dikembalikan dalam bentuk dua buah list yang diurutkan oleh H dan HT yang perlu interpreter. Cara pembacaan ada dimanual Plague, namun hal ini menyatakan bahwa hasil tidak bisa dilihat begitu saja oleh pengguna tetapi harus ada pemrosesan lebih lanjut.
3. Ada bagian dari Plague yang ditulis dalam bahasa Pascal, padahal implementasi bahasa C yang berkualitas baik lebih umum, sementara implementasi Pascal yang berkualitas baik sangatlah langka. Selain itu Plague juga bergantung pada beberapa utility UNIX dan GIVE sehingga menimbulkan masalah portability.

Aiken (1998) menciptakan perangkat lunak pendeteksi plagiarisme *source code* yang diberi nama Measure Of Software Similarity (MoSS) dalam publikasi di website *http://ftp.cs.berkeley.edu/~aiken/moss.html* tidak ada informasi yang diberikan mengenai laporan hasil deteksi sistem, tetapi pada paper yang tidak dipublikasikan oleh Goel dan Rao (2003), MoSS dibangun menggunakan algoritma *winnowing*.

Wise (1996) Menciptakan versi final dari YAP yaitu YAP3, tujuan dari pembuatan YAP adalah membuat suatu sistem deteksi plagiarisme yang dibangun dengan pondasi Plague dan mengatasi masalah-masalah yang dihadapi pengguna ketika menggunakan Plague, dengan memperkenalkan gagasan untuk penanganan substring yang dialihkan dan terbukti lebih efisien dari pada *Plague*. Gagasan tersebut adalah sebuah algoritma Running Karp-Rabin Greedy String Tiling. Seperti versi-versi sebelumnya pada YAP3 juga bekerja melalui 2 fase. Fase pertama tokenisasi tidak mengalami perubahan besar walaupun tokenizer bekerja dengan lebih baik dari pada tokenizer pada versi-versi sebelumnya. Selain itu, token di YAP3 bukan merupakan string melainkan numerik. YAP3 menggunakan algoritma Running Karp-Rabin Greedy String Tiling pada fase pembandingan. Kesamaan dari ketiga versi YAP yang telah dikembangkan terletak pada proses tokenisasi di fase pertama, dimana proses tokenisasi memiliki stuktur sebagai berikut :

1. Menghilangkan komentar dan *print-string* serta perubahan huruf besar ke kecil, menghilangkan huruf yang tidak ditemukan di identifier.
2. Mengganti sinonim kebentuk umum, contohnya pada bahasa pemrograman LISP second dan cadr diganti menjadi car dan cdr.
3. Mengidentifikasi blok fungsi atau procedure. Jika telah mengidentifikasi blok fungsi, cetak blok fungsi sesuai urutan pemanggilan
4. Pemanggilan terhadap fungsi atau procedure yang telah di-*expand* diganti dengan beberapa token yang merepresentasikan fungsi atau procedure itu.
5. Kenali dan cetak token yang merepresentasikan bagian dari bahasa pemrograman dari kamus data yang diberikan.

Prechelt, et al, (2000) dalam penelitiannya mengenai plagiarisme *source code* yang diimplementasikan pada perangkat lunak bernama JPlag. JPlag dibangun menggunakan algoritma Running Karp-Rabin Greedy String Tiling, bekerja malalui dua fase untuk melakukan deteksi plagiarisme. Pada fase pertama semua program di *parse* atau di *scan* terlebih dahulu kemudian dikonversi menjadi token string, setiap token mewakili satu baris *syntax* yang ada pada program. Pada fase kedua token string yang telah di-*generate* dibandingkan secara berpasangan

kemudian nilai *similaritas* setiap pasangan dihitung. Gambar 2.1 merupakan contoh *source code* Java dan kumpulan token yang mewakili setiap barisnya pada perangkat lunak JPlag.

Source Code Java	Token Hasil Generated

public class count {
public static void main(String[]args)
throws java.io.IOException {
int count = 0;
while (System.in.read() != -1)
count++;
System.out.println(count+" chars.");
}
}
	BEGINCLASS
	VARDEF,BEGINMETHOD
	
	VARDEF,ASSIGN
	APPLY,BEGINWHILE
	ASSIGN,ENDWHILE
	APPLY
	ENDMETHOD
	ENDCLASS

Gambar 2.1 Token Hasil *Generate* pada JPlag

Kustanto dan Liem (2009) melakukan penelitian tentang deteksi otomatis plagiarisme *source code* dengan membuat perangkat lunak yang diberinama Deimos, pada penelitiannya memakai algoritma Running Karp-Rabin Greedy String Tiling pada fase pembandingan, digunakan untuk mendeteksi *source code* pada pemrograman *Pascal* dan *LISt Processing* (LISP), butuh membangun sebuah *scanner* atau *parser* agar Deimos mampu mengenali bahasa pemrograman lain. Deimos juga bekerja dalam dua langkah, pertama *parsing source code* dan mengubahnya menjadi token kemudian langkah kedua membandingkan setiap pasangan token yang diperoleh pada langkah pertama.

Gambar 2.2 merupakan beberapa contoh daftar token untuk merepresentasikan *source code* bahasa pemrograman Pascal yang ada pada sistem Deimos.

Aturan Produksi	Token Integer	Keterangan
program identifier ; <block> .	354 168	Awal program Akhir program
<procedure declaration> ; <block> ;	797 833	Awal procedure Akhir
for identifier := <for_list> do <statement>	321 299	Awal statement for Akhir statement for
<variable> := <expression>	733	*Assigment*
Write ({ stringcon ,} <variable> {, <variable>});	775	Mencetak nilai ke layar
Read (<variable>);	496	Meminta input pengguna
Identifier = <type>	816	Definisi type

Gambar 2.2 Daftar token source code pascal pada sistem Deimos

Perbedaan pada penelitian yang dilakukan saat ini, terletak pada token yang dihasilkan untuk proses pembandingan, selain itu tentu saja pada teknik implementasinya termasuk bahasa pemrograman yang digunakan serta jenis bahasa pemrograman yang mampu dideteksi.

Untuk lebih jelasnya mengenai token yang dihasilkan dalam penelitian ini, akan dibahas pada bab IV sub bab 4.2. Daftar perbandingan pustaka diperlihatkan pada Tabel 2.1.

Tabel 2.1 Perbandingan penelitian lampau dengan sistem yang akan dibangun

Aspek	YAP3	JPlag	Deimos	CoPas Cek (Peneliti)
Algoritma Pembandingan	RKRGST	RKRGST	RKRGST	RKRGST
Dibangun dengan bahasa pemrograman	Java	Back end Java dan front end php	Back end Java dan front end php	VB.Net
Representasi source code yang dibandingkan	Token file, dengan setiap token berupa angka	Token string, dengan setiap token berupa string	Token string, dengan setiap token berupa angka	Token string, dengan setiap token berupa angka
Portability	*Cross-platform*	*Cross-platform*	*Cross-platform*	*Single Platform*
Type Aplikasi	Desktop based	Melalui antar muka web dan dieksekusi melalui aplikasi back end	Melalui antar muka web dan dieksekusi melalui aplikasi back end	Desktop based
Bahasa pemrograman yang mampu dideteksi	C, LISP	Java, C, C++ dan Scheme	Pascal, LISP	VB.6.0, VB.Net, C#, Delphi
Penanganan untuk bahasa pemrograman lain	Membangun tokenizer baru	Membangun tokenizer baru	Membangun tokenizer baru	Membangun tokenizer baru

BAB III
LANDASAN TEORI

3.1 Metode Deteksi Plagiarisme Source Code

Dua pendekatan utama yang telah dipakai untuk permasalahan plagiarisme *source code* yaitu *attribute-counting* dan *structure-based* (Clough, 2000). Pada metode *attribute-counting*, yang dibandingkan adalah ukuran kuantitatif beberapa matriks program, sedangkan pada metode *structure-based* yang dibandingkan adalah representasi dari stuktur program, misalkan representasi linier berupa *string, parse tree, data flow* dan lain-lain.

3.1.1 Metode attribute-counting

Pada sistem deteksi plagiarisme *source code* generasi awal menggunakan metode ini untuk membandingkan kemiripan antar kode program, setiap program mempunyai suatu angka yang merupakan suatu analisis kuantitatif dari beberapa fitur program (matriks) yang kemudian akan dibandingkan. Contoh matriks yang paling sederhana adalah ukuran program, misalnya jumlah *line of code, non blank non comment, execution statement.* Kemudian berbagai macam teknik terus dikembangkan agar lebih mampu mengenali berbagai modifikasi yang yang telah dilakukan, berbagai macam metode *attribute-counting* terus bermunculan, seperti menggunakan perhitungan jumlah *operator* dan *operands* oleh Halstead, metode *cyclomatic complexity* dari McCabe yang mengukur aliran kontrol program dengan menghitung jumlah *execution path.*

Dari metode yang telah dicipatakan berbagai matriks dikombinasikan sehingga terdapat beberapa angka yang akan mewakili setiap program. Program dianggap sama jika hampir semua atau semua angka tersebut sama, kemudian dilakukan *tuning* dari sistem tersebut dengan menentukan bobot untuk setiap parameter, sehingga matriks yang lebih penting akan lebih berpengaruh pada tingkat similaritas kedua program yang dibandingkan. Walaupun *tuning* telah dilakukan untuk memperbaiki metode ini, metode *attribute-counting* hanya berhasil melakukan deteksi dengan efektif pada modifikasi-modifikasi yang

dilakukan secara sederhana, metode tersebut masih belum mampu mendeteksi modifikasi *source code* yang lebih kompleks.

3.1.2 Metode structure-based

Metode *attribute-counting* tidak bisa menggambarkan struktur program secara baik untuk membedakan *source code* hasil plagiarisme dan *source code* yang bukan merupakan hasil plagiarisme (Faidhi dan Robinson, 1987). Kemudian dikembangkan metode berbasis struktur agar lebih bisa menggambarkan struktur program secara baik dan melakukan pengecekan terhadap modifikasi yang lebih kompleks, contohnya mengenali bagian kode program yang isinya diambil sebagian dari kode sumber aslinya, perubahan *statement*, perubahan urutan *statement*, menambah komentar atau mengubah komentar yang ada dan modifikasi-modifikasi lainnya.

Metode ini tidak membandingkan representasi kuantitatif dari atribut program seperti pada *attribute-counting*, melainkan membandingkan suatu representasi dari struktur program. Secara umum proses pendeteksian menggunakan metode berbasis struktur terbagi menjadi dua tahap yaitu :

1. *Tokenization* yaitu *parsing* kode program menjadi kumpulan token. Tokenization diimplementasikan pada proses *scanning* dan *parser*[1].
2. Membandingkan setiap procedure yang terdapat pada *source code* A dengan setiap procedure yang ada pada *source code* B.

Pada sistem yang dirancang, suatu procedure dianggap memiliki kesamaan jika jumlah string yang tertandai mencapai lebih dari atau sama dengan nilai sensitifitas deteksi yang telah ditentukan oleh user pengguna aplikasi.

3.2 Algoritma Pencocokan String

Pada sistem yang dibangun, proses pengecekan *syntax* untuk menemukan bagian-bagian yang sama pada *source code* program tidak terlepas pada masalah *string matching*. Pada sub bab ini, sebelum berbicara mengenai algoritma Running Karp-Rabin Greedy String Tiling (RKRGST) akan dibahas terlebih dahulu algoritma Greedy String Tiling yang telah di *tuning*.

---

[1] Parser bertanggung jawab untuk melakukan analisis sintaks, yang terkait dengan grammar atau aturan kalimat pada suatu bahasa pemrograman.

Ada beberapa istilah yang perlu diketahui sebelum algoritma Greedy String Tiling (GST) dan Algoritma RKRGST dapat dijelaskan.

a. ***Token, token string* dan *mark*.**

- *Token* merupakan deretan *string* dengan panjang tertentu, *Token* merepresentasi elemen-elemen yang ada pada *source code*.
- *Token string* merupakan *array* yang berisi token bukan *array* yang berisi karakter
- *Mark* merupakan sebuah penanda terhadap karakter yang ada pada string. Jika suatu karakter yang ada pada *string* telah di tandai (*mark*), maka mestilah karakter tersebut telah merupakan bagian dari suatu *tile*.

b. ***Maximal-match***

Sebuah *maximal-match* menyimpan posisi dan panjang sebuah *substring* yang sama pada kedua *string* yang dibandingkan. Cara membentuk *maximal-match* yaitu ketika pasangan *token* yang sama ditemukan, kemudian karakter pada posisi berikutnya ikut dicocokkan, proses pencocokan akan berhenti apabila salah satu dari tiga kondisi berikut ditemui :

1. Jika pasangan karakter yang tidak cocok ditemukan
2. Jika ditemukan karakter yang telah ditertandai (*marked*)
3. Jika tidak ada karakter yang bisa dibandingkan lagi pada salah satu *string* (akhir *string* dicapai)

Notasi *maximal-match* terdapat di fungsi *scanpattern* (Gambar 3.2 baris ke-21) yaitu *match*(p,t,k), dengan p dan t merupakan posisi karakter pertama yang memiliki kemiripan pada *text string* dan *pattern string* dan k merupakan panjang karakter yang cocok pada *substring* tersebut. *Maximal-match* bersifat temporer dan ada kemungkinan bukan sesuatu yang tunggal, karena karakter-karakter yang ada disuatu *substring* yang menjadi bagian dari suatu *maximal-match* bisa saja merupakan bagian dari *maximal-match* yang lainnya.

c. ***Tile***

Sebuah *tile* menyimpan posisi dan panjang sebuah *substring* yang sama pada kedua *string* yang dibandingkan. *Tile* dibentuk dari *list of maximal-match*. Dalam proses membentuk *tile* dari sebuah *maximal-match*, karakter yang terdapat pada

token yang merupakan elemen dari *maximal-match* tersebut ditandai (*marked*). Bedanya dengan *maximalmatch*, *tile* bersifat permanen dan unik, karena karakter yang terdapat pada *substring* yang menjadi bagian dari suatu *tile* tidak akan menjadi bagian *tile* yang lain karena telah tertandai.

d. Minimum-match-length

Pada proses pencocokan terkadang didapat *maximal-match* yang ukurannya pendek. *Maximal-match* yang berukuran pendek tersebut dapat diabaikan karena dianggap memiliki kemiripan yang tidak signifikan. *Maximal-match* dengan panjang satu atau dua karakter saja tidak akan berarti ketika membandingkan dua buah *source code*. Karena itu, ditentukan sebuah nilai yang disebut *minimum-match-length*. Pada fase markstring, semua *maximal-match* yang mempunyai panjang di bawah *minimum-match-length* tidak akan dibentuk menjadi sebuah *tile*.

3.2.1 Algoritma Greedy String Tiling (GST) hasil *tuning*

Tujuan dari algoritma ini adalah membentuk *tile* dari kedua *text string* tanpa adanya *overlapping* dengan memaksimalkan jumlah karakter yang ditandai. Tanpa *overlapping* yang dimaksud adalah sebuah karakter yang telah tertandai pada *text string* mestilah merupakan elemen dari satu buah *tile* saja.

Maximal-match yang memiliki panjang lebih besar akan lebih diutamakan untuk membentuk sebuah *tile* dari pada *maximal-match* yang lebih pendek, karena *maximal-match* yang lebih panjang, lebih mencerminkan kecocokan dan kecil kemungkinan bahwa kecocokan tersebut hanyalah kebetulan saja.

Biasanya *text string* yang dipilih untuk diuji adalah *text string* yang memiliki panjang karakter yang lebih sedikit dari pada yang terdapat pada *text string* pembandingnya.

Kasus terburuk algoritma GST mempunyai kompleksitas $O(n^3)$. Oleh sebab itu (Wise, 1993), melakukan *tuning* terhadap algoritma GST untuk memperbaiki kompleksitas rata-rata algoritma ini. *Pseudocode* algoritma GST yang telah di *tuning* digambarkan pada Gambar 3.1.

```
function TunedGreedyStringTiling (P[1..M],T[1..N] : tokenString, minimum_match_length:
integer) integer
(membandingkan setiap karakter pada P dan T)
(text string P[1..M] merupakan text string yang lebih pendek dibandingkan
dengan text string T[1..N])
(jika ditemukan sederetan string yang sama dan panjangnya lebih
besar dari minimum-match-length, maka string tersebut akan ditandai)
(mengembalikan jumlah string yang telah ditandai)
KAMUS
length_of_tokens_tiled : integer (jumlah string yang sudah ditandai)
maxmatch : integer (search-length)
p: integer (posisi index pada pattern string)
t: integer (posisi index pada text string)
match_list : list of match (List yang berisi maximal-matches)
j,k : integer
ALGORITMA
length_of_tokens_tiled ← 0
repeat
  /* mulai fase scanpattern */
  for each unmarked(P[p]) in P[0..M] do   {tuning ke 1}
      if distance from p to next tile ≤ minimum_match_length then
        p ← position of first unmarked token after next tile   {tuning ke 2}
      else
        maxmatch ← minimum_match_length
        for each unmarked(T[t]) in T[0..N] do   {tuning ke 3}
            if P[p] = T[t] then
{tuning ke 4}  if p+maxmatch-1<M and t+maxmatch-1<N and
                  hash-value(P[p..p+maxmatch-1]) = hash-value(T[t..t+maxmatch-1]) and
                  unmarked(P[p+maxmatch-1]) and unmarked(T[t+maxmatch-1]) then
                  for j← maxmatch-1 downto 1    {tuning ke 5}
                      if P[p+j] = T[t+j] then
                        k ← maxmatch
                        while p+j<M and t+j<N and P[p+k]=T[t+k] and
                              unmarked(P[p+k]) and unmarked(T[t+k]) do
                              k ← k+1
                  if k = maxmatch then
                    add match(p,t,k) to match_list
                  else if k > maxmatch then
                    restart match_list, add match(p,t,k) to match_list
                    maxmatch ← k
      for each match(p,t,maxmatch) in match_list do
          if all tokens in match(p,t,maxmatch) are unmarked then
            for j ← 0 to maxmatch-1 do
              markToken(P[p+j])
              markToken(T[t+j])
            length_of_tokens_tiled ← length_of_tokens_tiled + maxmatch
until maxmatch = minimum-match-length
→ length_of_tokens_tiled
```

Gambar 3.1 *Pseudocode* algoritma Greedy String Tiling hasil *tuning* (Wise, 1993)

Beberapa *tuning* yang dilakukan pada algoritma GST diantaranya adalah :

1. Pada *pattern string*, hanya karakter-karakter yang belum tertandai saja yang akan dibandingkan.
2. Pada *pattern string*, jika jarak dari posisi indeks saat itu ke *tile* berikutnya lebih kecil dari *minimum-match-length*, maka pada karakter yang belum tertandai pada *pattern string* mulai dari posisi indeks yang ada saat itu akan diabaikan beserta *tile* yang mengikutinya. Posisi indeks akan bergeser ke karakter pertama yang belum tertandai yang letaknya setelah *tile* tersebut.
3. Pada *text string* hanya karakter-karakter yang belum tertandai saja yang akan dibandingkan dengan karakter yang terdapat pada *pattern string*.
4. Setelah karakter yang sama ditemukan pada posisi p pada *pattern string* dan t pada *text string*, terdapat posisi pembandingan berikutnya yang lebih baik daripada p+1 dan t+1. Posisi tersebut yaitu pada p+maxmatch-1 dan t+maxmatch-1, dengan menghitung nilai *hash* untuk kumpulan karakter yang terdapat pada *pattern string* mulai dari posisi p+1 sampai p+maxmatch-1 dan pada *text string* mulai dari posisi t+1 sampai t+maxmatch-1 serta pastikan bahwa karakter yang terdapat diposisi p+maxmatch-1 dan t+maxmatch-1 belum tertandai. Jika kondisi tersebut terpenuhi, lakukan pembandingan terhadap nilai *hash* nya.
5. Jika nilai *hash* pada p+maxmatch-1 dan t+maxmatch-1 memiliki kesamaan maka pembandingan karakter per karakter dilakukan mundur mulai dari posisi indeks p+maxmatch-1 sampai p+1 yang ada pada *pattern string* dan posisi indeks t+maxmatch-1 sampai t+1 pada *text string*, karena ketidak cocokan karakter yang terdapat pada kedua *substring* seringkali terjadi pada posisi akhir dari *substring*.

3.2.2 Algoritma Running Karp-Rabin Greedy String Tiling (RKR-GST)

Algoritma Running Karp-Rabin Greedy String Tiling pertama kali dipakai pada sistem *Neweyes*, yaitu sebuah sistem untuk *alignment biosequences* nukleotida dan asam amino (Wise, 1993). Salah satu masalah yang dihadapi oleh sistem *Neweyes* tersebut yaitu menentukan *similarity* antara dua buah *string*.

Masalah tersebut sama dengan yang dihadapi oleh sistem deteksi plagiarisme, sehingga Michael J. Wise juga menerapkan algoritma ini pada aplikasi YAP3. (Wise, 1996). Mengatakan algoritma Running Karp-Rabin Greedy String Tiling merupakan pengembangan lebih lanjut dari algoritma Greedy String Tiling dengan menerapkan algoritma Karp-Rabin pada fase *scanpattern*.

Algoritma Karp-Rabin adalah salah satu algoritma pencarian string yang di buat oleh Michael Rabin dan Richard Karp pada tahun 1987, algoritma ini menggunakan fungsi *hash* pada proses pencarian string yang dicari dengan substring pada teks. Suatu string yang dicari memiliki kesamaan apabila *hash value* keduanya sama. jika kedua *hash value* sama maka akan dilakukan perbandingan sekali lagi terhadap karakter-karakternya, apabila *hash value* tidak sama maka substring akan bergeser ke kanan. Perhitungan nilai *hash* untuk pembandingan itulah yang diterapkan pada algoritma Running Karp-Rabin Greedy String Tiling ini.

3.2.2.1 Top level algoritma RKR-GST

Top level algoritma RKR-GST menggambarkan langkah-langkah untuk menemukan dan menandai string yang cocok diantara teks yang dibandingkan. Procedure ini sangat mirip dengan algoritma GST, Perbedaannya terletak pada fungsi scanpattern yang didalamnya terdapat algoritma Running Karp-Rabin.

Pada *pseudocode* Top level algoritma RKRGST, nilai *search length* (*s*) didefinisikan terlebih dahulu, ini yang membedakan algoritma ini dengan algoritma Greedy String Tiling. Pada top level algoritma RKRGST putaran *Loop* pertama dimulai dengan memanggil fungsi *scanpattern* dengan nilai *s* yang telah didefinisikan diawal. Fungsi *scanpattern* mencari *substring* yang sama dimulai dengan panjang *s* dan akan mengembalikan nilai *s* terbesar setelah keluar dari *scanpattern*, jika nilai *s* lebih besar dari dua kali nilai awal *s* maka ulangi untuk melakukan *scanpattern* dengan nilai *s* yang baru, tetapi jika *s* tidak lebih besar dari dua kali panjang nilai awal *s* maka menuju *markstring* untuk melakukan penandaan pada *list-of-maximal-match* yang telah didapat dari fungsi *scanpattern*. *Pseudocode* Top Level algoritma RKR-GST digambarkan pada Gambar 3.2.

```
procedure RunningKarpRabinGreedyStringTiling(P[0..M],T[0..N] : teksString,
minimum_match_length : integer)
(teks string P[1..M] merupakan teks string yang lebih pendek dibandingkan
dengan teks string T[1..N])
(mencari substring yang bernilai sama dan panjangnya lebih besar dari
minimum_match_length)
KAMUS
s : integer (search-length), Lmax : integer, stop : Boolean
    ALGORITMA
1   s ← 4  /* inisialisasi panjang search-length */
2   stop ← false
3
4   repeat
        Lmax ← scanpattern(s)    /* Lmax = maximal-matches terbesar */
                                 /* yang terdapat pada iterasi tersebut */
5       if Lmax>(2*s) then   /* string yang ditemukan sangat panjang */
6           s ← Lmax   /* jangan tandai tile, coba scanpattern dengan nilai s yang besar */
7       else
8           markstrings(s)  /* ciptakan tile dari list-of-maximal-match yg terdapat queues */
9           if s> (2*minimum_match_length) then
10          s ← s div 2
11          else if s > minimum_match_length then
12          s ← minimum_match_length
13          else
14          stop ← true
15  until stop
```

Gambar 3.2 *Pseudocode* Top-level Algoritma RKR-GST (Wise, 1993)

3.2.2.2 Fase *Scanpattern*

Gambar 3.3 merupakan *pseudocode* fase *scanpattern*, yang bertujuan mencari *substring* yang sama sepanjang *search length (s)* yang terdapat pada TeksStringT dan TeksStringP berdasarkan nilai *hash* nya. Hasil pencarian *substring* yang sama disebut *maximal-match* dan akan disimpan kedalam *list-of-maximal-match*. Pada TeksStringT untuk setiap karakter yang belum tertandai dengan panjang *s* akan dibentuk token, misalkan untuk TeksStringT $T_{[t \ldots t+s-1]}$ dengan t antara 1 sampai [t-s] *token* dibentuk dan dihitung *hash-value* dari masing-masing *token* tersebut.

Jika pada TeksStringT telah terdapat karakter yang tertandai, maka cek jarak dari posisi indeks saat itu ke *tile* yang ada didepannya. Jika lebih kecil dari *search-length (s)*, maka bagian yang belum tertandai pada TeksStringT mulai dari posisi indeks tersebut akan diabaikan, beserta *tile* yang mengikutinya. Posisi indeks akan bergeser ke karakter pertama yang belum ditandai yang letaknya setelah *tile* tersebut, setelah seluruh TeksStringT yang belum tertandai telah dibentuk token dan dihitung *hash value* dari masing-masing token, maka kumpulan token dari TeksStringT tersebut, disebut sebagai *hashtable*.

Pada TeksStringP proses yang sama juga dilakukan, dibentuk token dari TeksStringP $P_{[p..p+s-1]}$ yang belum tertandai sepanjang *search-length (s)*, tetapi bedanya untuk setiap satu token yang telah terbentuk pada TeksStringP, hitung *hash value* token tersebut, kemudian langsung dibandingkan *hash value* tersebut dengan *hash value* yang ada di *hashtable* yang terbentuk dari TeksStringT.

Jika ditemukan adanya *hash-value* yang sama, maka lakukan pengecekan terhadap karakter yang berada di posisi berikutnya seperti pada algoritma GST, sampai salah satu dari tiga kondisi berhenti berikut ditemui :

1. Jika pasangan karakter yang tidak cocok ditemukan
2. Jika ditemukan karakter yang telah ditertandai
3. Jika tidak ada karakter yang bisa dibandingkan lagi pada salah satu *string* (akhir dari *string* dicapai).

```
function scanpattern (P[0..(M-(s-1))], T[0..(N-(s-1))] : tokenstring, s: integer) → integer
(membandingkan setiap token pada P dan T)
(token string P[1..M-s] dengan jumlah token string yang lebih pendek dibandingkan
dengan token string T[1..N-s])
(mencari pasangan substring yang bernilai sama dan panjangnya lebih besar dari search length s)
KAMUS
p: integer (posisi indeks token pada pattern string)
t: integer (posisi indeks token pada text string)
maxmatch_list : double-linked list of queues of match
hashtable : array of integer (menyimpan nilai hash text string), k : integer
1   for each unmarked(T[t]) in T[0..N] do
2       if distance from t to next tile ≤ s then
3           t ← position of first unmarked token after next tile
4       else
            create the Karp-Rabin hash-value(T[t..t+s-1])
5           add hash-value(T[t..t+s-1]) to hashtable
6   for each unmarked(P[p]) in P[0..M] do
7       if distance from p to next tile ≤ s then
8           p → position of first unmarked token after next tile
9       else
10          create the Karp-Rabin hash-value(P[p..p+s-1])
            /* cek hashtable, cari hash value yang sama */
12          for each hashtable entry = hash-value(P[p..p+s-1]) do
13              k ← s
14              while p+k<M and t+k<N and P[p+k]=T[t+k] and
15                    unmarked(P[p+k]) and unmarked(T[t+k]) do
16                    k ← k+1
17              if k > (2*s) then  /*maximal-match sangat panjang, tinggalkan scanpattern*/
18                    /* restart s-k di top-level algoritma */
19                    → k
20              else
21                    add match(p,t,k) to maxmatch_list
22  → highest value in maxmatch_list
```

Gambar 3.3 *Pseudocode* fase *scanpattern* RKR-GST (Wise, 1993)

Pasangan substring yang sama akan dikonversi menjadi *maximal-match* seperti pada algoritma GST. Jika ditemukan panjang *maximal-match* lebih besar dari dua kali panjang *s*, sehingga menghasilkan *maximal-match* yang sangat panjang maka kembali pada top level algoritma dengan memanggil fungsi *scanpattern* dengan nilai *s* yang baru. Tetapi jika *maximal-match* yang ditemukan tidak lebih besar dari atau sama dengan dua kali panjang *s* maka tambahkan *maximal-match* tersebut kedalam *list-of-maximal-match*.

Struktur data yang digunakan untuk menyimpan *maximal-matches* pada *pseudocode* tersebut adalah sebuah *doublelinked-list of queues* tetapi pada penelitian ini peneliti menggunakan struktur data berupa *array structure*.

3.2.2.3 Fase Markstring

Fase *Markstring* bertujuan melakukan penandaan pada *list-of-maximal-match* yang telah didapat dari fase *scanpattern*, *list-of-maximal-match* sebelumnya telah diurutkan secara *descending* sehingga penandaan dimulai pada *maximal-match* yang lebih besar. Sebelum dilakukan penandaan pada token string akan dilakukan pengecekan kembali apakah karakter-karakter tersebut benar-benar sama dan belum tertandai. Jika kondisi tersebut terpenuhi maka sebuah *tile* dapat dibentuk dari *maximal-match* tersebut.

Minimum-match-length merupakan parameter yang dapat merepresentasikan sensitivitas algoritma ini, semakin rendah nilai *minimum-match-length* maka sensitifitas deteksi semakin tinggi, ini berarti suatu *source code* yang berbeda kemungkinan dapat terdeteksi memiliki suatu kesamaan yang signifikan, dan apabila sensitifitas deteksi tinggi maka segmen-segmen yang sama pada *source code* yang nilai *maximal-match* nya dibawah nilai sensitifitas yang telah ditentukan maka akan dianggap sebagai *source code* yang tidak memiliki kesama.

Mengenai cara kerja algoritma Running Karp-Rabin Greedy String Tiling dalam melakukan proses pencocokan string untuk lebih jelasnya dapat dilihat pada halaman lampiran. *Pseudocode* fase *markstrings* digambarkan pada Gambar 3.4.

```
procedure markstrings (P[0..M],T[0..N] : tokenString, s: integer)
(membandingkan setiap token pada maximal-matches yang telah didapatkan pada
fase scanpattern)
(token string P[1..M] merupakan token string yang lebih pendek dibandingkan dengan token
string T[1..N])
KAMUS
p: integer (posisi indeks token pada pattern string)
t: integer (posisi indeks token pada text string)
maxmatch_list : double-linked list of queues of match
                (List berisi maximal-match)
L: integer (ukuran maximal-match pada queue di iterasi tersebut)
    ALGORITMA
    /* dimulai dari antrian pertama pada list-of-maximal-match */
1   while there is a non-empty queue do
2       if the current queue is empty then  /* maximal-match dengan ukuran < MML */
3         drop to next queue
4       else
5           if all tokens in match(p,t,L) are unmarked then
6             if P[p+j]=T[t+j] for all j← 0 to s-1 then
7               for j←0 to L-1 do
8                   mark_token(P[p+j])
9                   mark_token(T[t+j])
10              length_of_tokens_tiled ← length_of_tokens_tiled + L
11          else if (L – length of marked tokens in match(p,t,L)) ≥ s then
12              add unmarked portion to maxmatch_list
13          delete match(p,t,L) from queue
```

Gambar 3.4 *Pseudocode* fase *markstrings* RKR-GST (Wise, 1993)

Pada sistem yang dibangun, peneliti menggunakan input parameter sensitifitas deteksi menggunakan nilai berupa persen (%), yang artinya setelah pencocokan string pada procedure tersebut telah selesai dilakukan, kemudian akan dihitung jumlah string yang tertandainya. Jika persentase string yang tertandai pada procedure tersebut mencapai lebih besar atau sama dengan nilai sensitifitas deteksi yang telah ditentukan oleh pengguna, maka *procedure* tersebut dianggap memiliki kesamaan terhadap *procedure* pada *soure code* pembandingnya.

3.3 Hashing

Hashing adalah suatu cara untuk mentransformasikan sebuah string menjadi suatu nilai yang disebut dengan nilai *hash* (*hash value*) dengan panjang tertentu, dimana nilai tersebut berfungsi sebagai penanda dari string tersebut. Fungsi untuk menghasilkan nilai ini disebut fungsi *hash*, sedangkan nilai yang dihasilkan disebut nilai *hash*. Untuk kata *w* dengan panjang *n* maka nilai *hash (w)* dapat didefinisikan seperti pada Gambar 3.5.

$$hash\ (w) = w_1 + w_n + (n * z)$$

Gambar 3.5 Perhitungan nilai *hash*

Keterangan :

w_1 = Kode ASCII karakter pertama pada string w

w_n = Kode ASCII karakter terakhir (ke- n) pada string w

n = Jumlah karakter pada string w

z = Banyak macam karakter yang digunakan (A-Z) + (0-9)

Fungsi *hash* ini akan diimplementasikan untuk menghitung nilai *hash* pada *token* yang memiliki panjang karakter sejumlah *search length* (s) yang terdapat di fase *scanpattern* pada algoritma RKRGST.

Algoritma Running Karp-Rabin Greedy String Tiling didasarkan pada fakta jika dua buah string sama maka *hash value* nya pasti sama, sebaliknya *hash value* sama tidak menjamin stringnya akan sama, tetapi kemungkinan besar dia sama. Untuk itu pada fase *markstring* akan dilakukan pengecekan kembali terhadap setiap karakternya sebelum dilakukan penandaan pada string tersebut.

3.4 Dice's Coefficient

Untuk mengukur nilai similaritas terhadap 2 (dua) *source code* yang dibandingkan, penulis menggunakan metode Dice's Coefficient yang dapat dinyatakan dengan persamaan (3.1).

$$s = \frac{2 * n}{t1 + t2} \qquad (3.1)$$

Keterangan :

n = jumlah procedure yang sama

t1 = jumlah procedure yang ada di modul *source code* 1

t2 = jumlah procedure yang ada di modul *source code* 2

BAB IV
ANALISIS DAN RANCANGAN SISTEM

Berdasarkan pada permasalahan yang telah dituangkan pada BAB I serta pemahaman mengenai teori dan konsep sebagai dasar pemikiran untuk menyelesaikan permasalah dengan melakukan analisis dan perancangan sistem. Selanjutnya sebuah program aplikasi akan dibangun berdasarkan rancangan yang telah disusun dan akan digunakan untuk melakukan pengujian.

Sistem yang dirancang merupakan sistem yang diharapkan dapat mengecek kemiripan dari 2 (dua) *modul source code* yang akan dibandingkan dengan teknik *string matching* menggunakan algoritma Running Karp-Rabin Greedy String Tiling, serta mengukur persentase kemiripannya menggunakan metode dice coefficient.

4.1 Penanganan Sistem Terhadap Modifikasi-Modifikasi Source Code

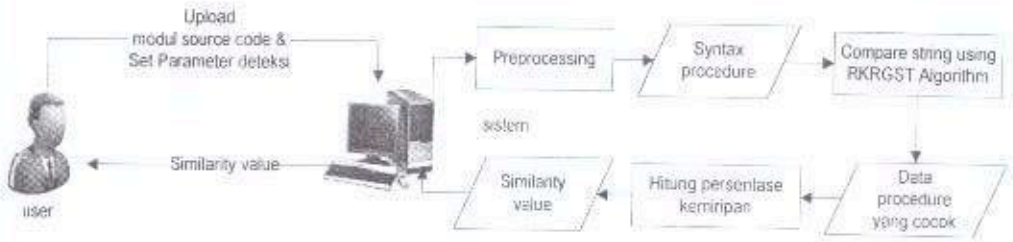
Untuk mengenali modifikasi-modifikasi yang dilakukan terhadap *source code*, guna menghindari tindakan plagiarisme maka pada sistem dirancang beberapa cara penanganan yang dapat dilihat pada Tabel 4.1.

Tabel 4.1 Penanganan terhadap modifikasi-modifikasi yang dilakukan

No	Jenis Modifikasi	Cara penanganan
1	Mengubah nama *procedure/function/method*	Mengabaikan nama *procedure / function/method*
2	Mengubah nama *variable*	Mengabaikan nama *variable*
3	Mengubah tipe data *variable* dengan tipe data lain	Mengabaikan seluruh tipe data *variable*
4	Mengubah atau menambah komentar	Mengabaikan seluruh komentar
5	Mengubah pemanggilan suatu *procedure* dengan mengganti nama *procedure* yang dipanggil	Mengabaikan pemanggilan suatu *procedure*
6	Mengubah urutan *statement*.	Algoritma RKR-GST mampu menemukan bagian-bagian yang sama, tanpa terpengaruh letak posisi *substring*
7	Menyalin sebagian *source code* atau hanya sebagian isi *procedure*	

4.2 Arsitektur Sistem

Secara umum sistem yang dirancang meliputi tahap *preprocessing* dan proses pengecekan kemiripan serta perhitungan kemiripan kedua pasangan *source code*. Desain arsitektur sistem dapat dilihat pada Gambar 4.1.



Gambar 4.1 Arsitektur sistem secara keseluruhan

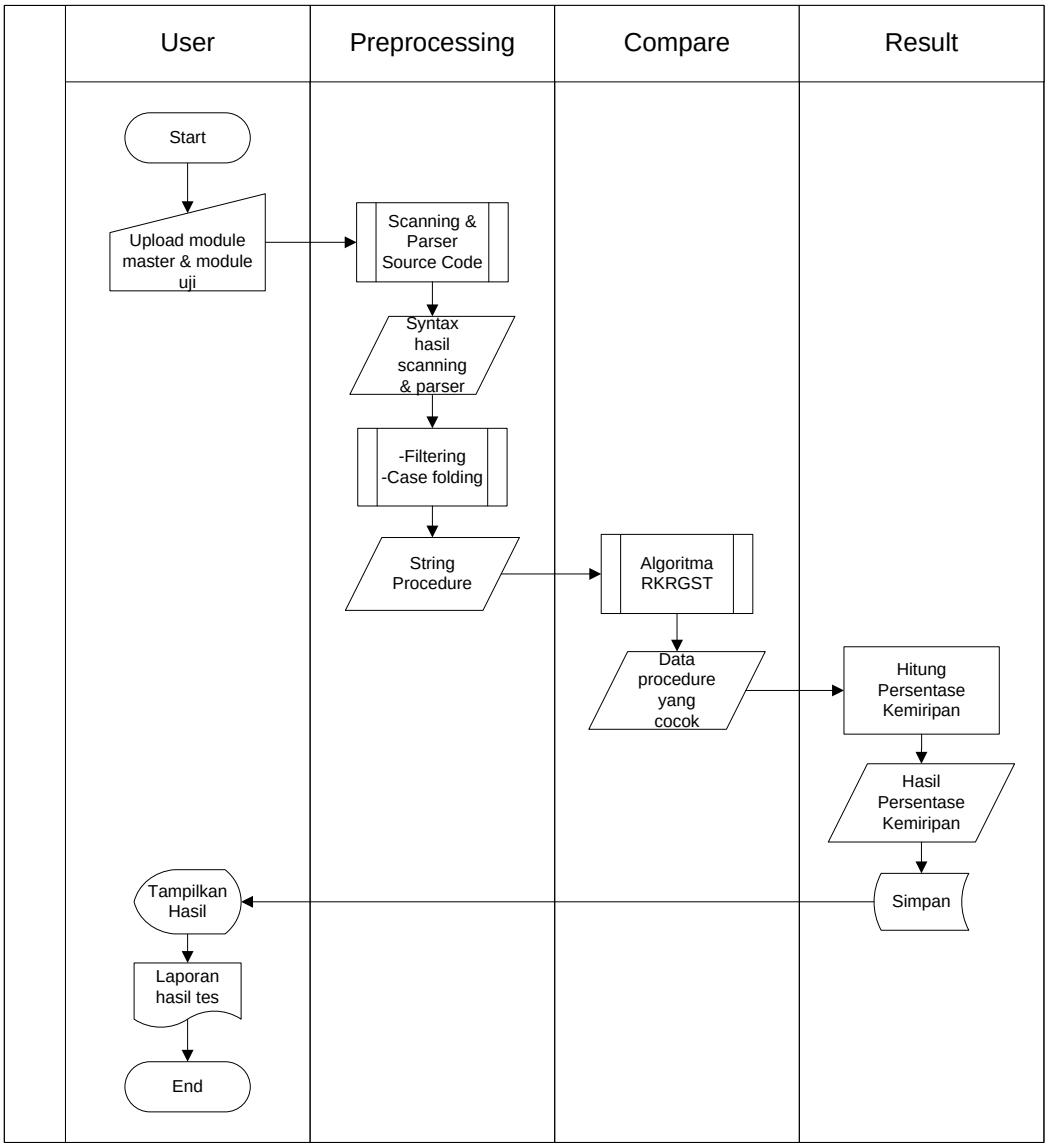
Secara garis besar sistem yang akan dirancang bertujuan untuk mendeteksi tingkat kemiripan antar dua modul *source code* hasil pembandingan. Input sistem berupa *source code* dari beberapa bahasa pemrograman yang telah disebutkan pada batasan masalah.

Diawal penggunaan aplikasi, user melakukan upload modul *source code* ke dalam sistem, kemudian sistem melakukan *preprocessing* terhadap *source code*. Proses *preprocessing* yang dilakukan meliputi *scanning* dan *parser* untuk mendapatkan *syntax* procedure, langkah selanjutnya untuk setiap *syntax procedure* yang ada, akan dilakukan proses *filtering variable*, *filtering stop word* dan *filtering* tanda baca, kemudian dilakukan proses *case folding* yaitu mengubah seluruh karakter yang ada pada *syntax procedure* menjadi huruf kecil.

Seluruh *syntax procedure* yang telah melewati tahap *preprocessing* akan dilakukan proses pengecekan kemiripan terhadap *syntax procedure* yang ada pada *source code* pembandingnya menggunakan algoritma RKRGST. Cara kerja dari algoritma RKRGST dapat dilihat pada halaman lampiran.

Pada aplikasi yang dirancang suatu *procedure* akan dianggap sama jika jumlah string yang tertandai pada *procedure* yang sedang dibandingkan mencapai nilai yang lebih besar atau sama dengan nilai sensitifitas deteksi yang telah ditentukan oleh pengguna aplikasi. Sebagai contoh ketika pengguna melakukan seting

sensitifitas deteksi 80% artinya setiap satu *procedure* yang dibandingkan akan dianggap memiliki kesamaan jika jumlah string tertandainya mencapai 80% dari total panjang string yang terdapat pada *procedure* tersebut. Flowchart sistem secara keseluruhan digambarkan pada Gambar 4.2.

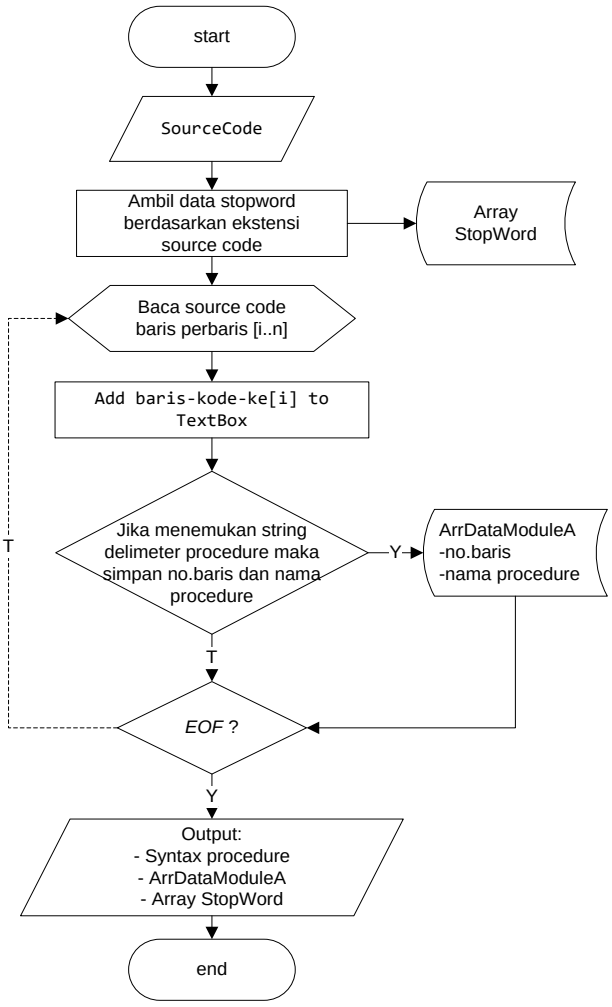


Gambar 4.2 Flowchart sistem secara keseluruhan

4.3 Sub Proses Sub Proses yang Terdapat pada Sistem

4.3.1 *Scanning* dan *parser source code*

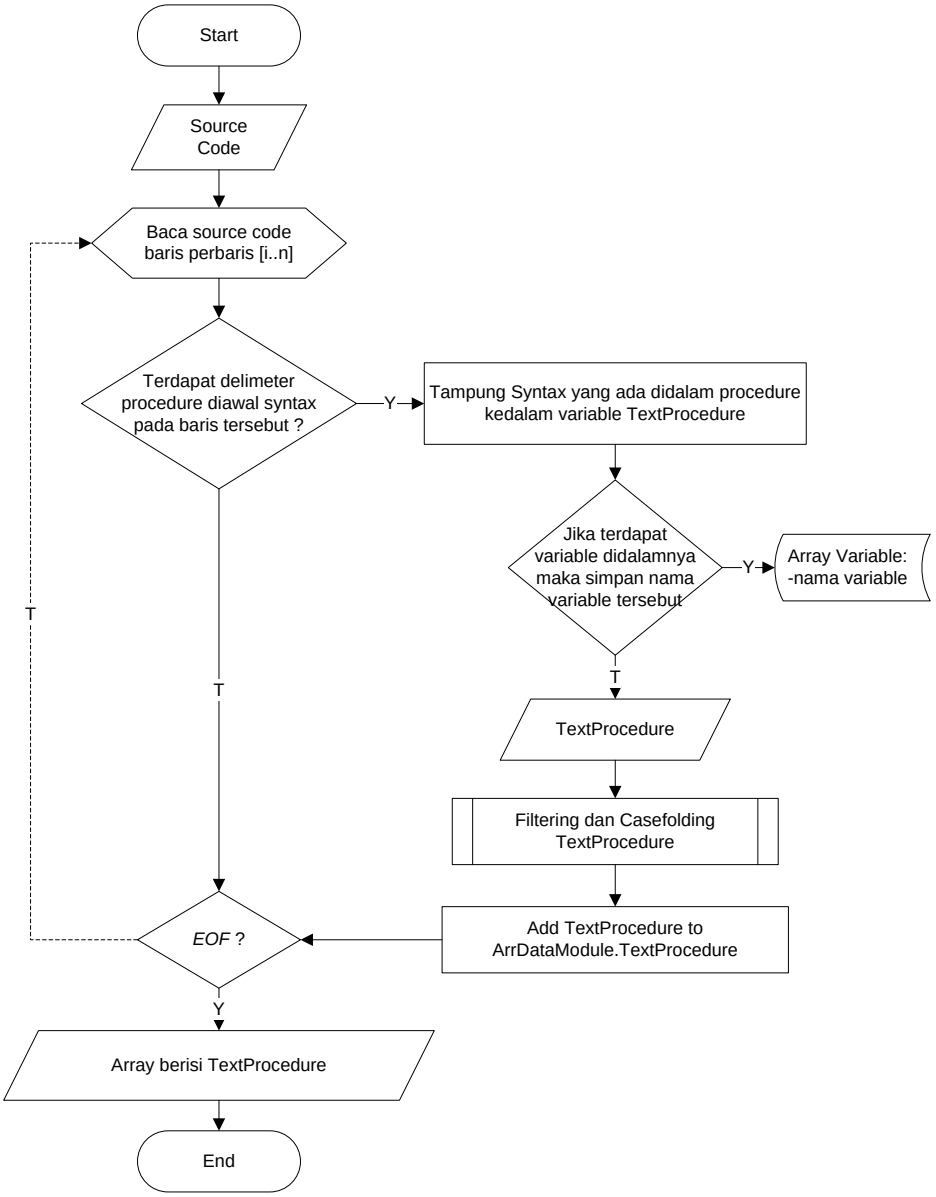
Scanning dan *parser* pada tahap ini merupakan proses baca modul yang bertujuan mendapatkan *syntax* program yang terdapat pada modul *source code* dan akan diambil data *stopword* berdasarkan ekstensi dari *source code* yang diupload. Pembacaan *source code* dilakukan baris perbaris, pada setiap baris yang dibaca akan disalin kedalam sebuah *textbox,* kemudian dilakukan pengecekan terhadap *syntax* program dibaris tersebut apakah terdapat *delimeter procedure* diawal *syntax* pada baris kode tersebut. Jika kondisi itu terpenuhi maka simpan nama dari *procedure* tersebut beserta nomor baris-nya. Flowchart dari proses *scanning* dan *parser* modul *source code* digambarkan pada Gambar 4.3.



Gambar 4.3 Flowchart Scanning dan Parser Source Code

4.3.2 Proses *scanning* dan *parser* dalam mengenali struktur procedure

Gambar 4.4 adalah flowchart proses *scanning* dan *parser* dalam mengenali struktur *procedure* yang dilakukan dengan menganalisa *syntax* yang saat itu sedang di *scan.* Pada setiap baris kode yang di *scan* jika *syntax* pertama pada baris kode tersebut mengandung *string delimeter procedure*, maka *syntax* yang ada didalam procedure tersebut akan disimpan kedalam suatu variable yang diberinama “TextProcedure”.

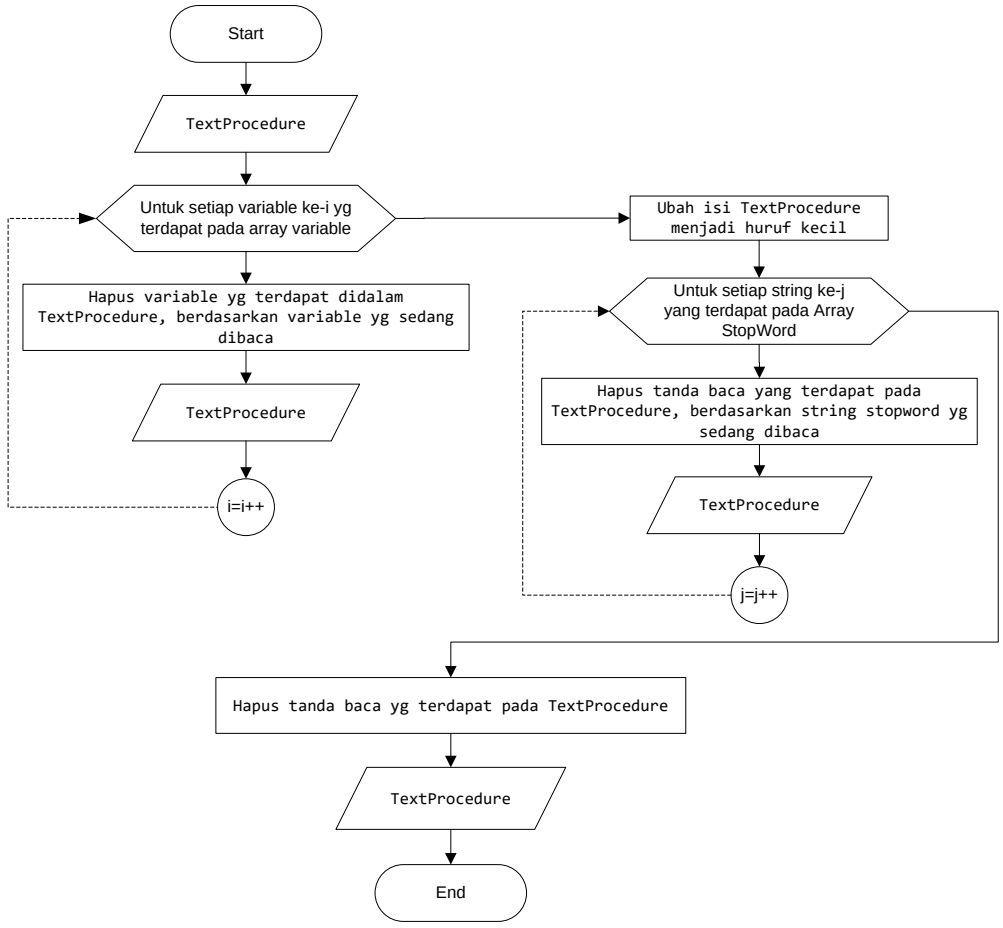


Gambar 4.4 Flowchart Scanning dan Parser dalam mengenali struktur procedure

Untuk setiap syntax procedure yang telah ditampung kedalam variable *TextProcedure* akan langsung dilakukan proses filtering dan casefolding, sehingga akan menghasilkan syntax procedure yang mewakili isi dari procedure tersebut untuk selanjutnya dilakukan proses pengecekan menggunakan algoritma RKRGST.

4.3.3 *Filtering* dan *casefolding*

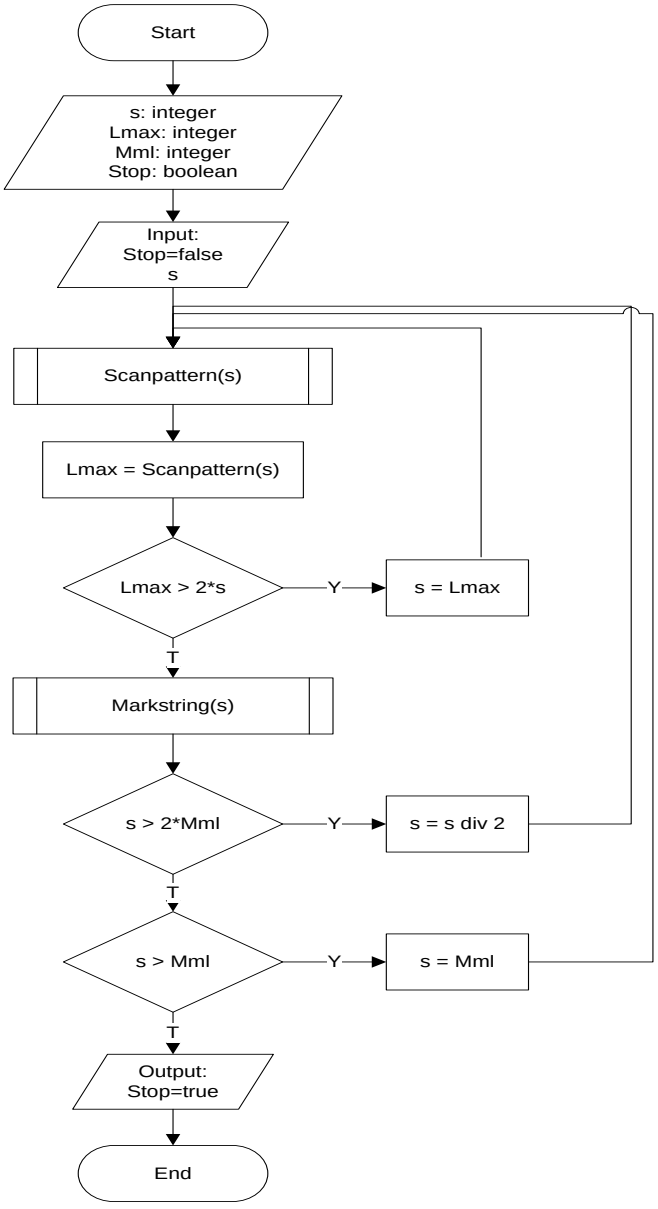
Terdapat 3 tahapan dalam proses filtering yaitu filtering keberadaan variable yang terdapat pada *TextProcedure* sehingga terbebas dari variable, filtering keberadaan *stopword* serta filtering keberadaan tandabaca, selanjutnya diikuti proses merubah *text procedure* menjadi huruf kecil (*casefolding*). Flowchart *filtering* dan *casefolding* digambarkan pada Gambar 4.5.



Gambar 4.5 Flowchart Proses Filtering dan Casefolding

4.3.4 Top level algoritma RKRGST

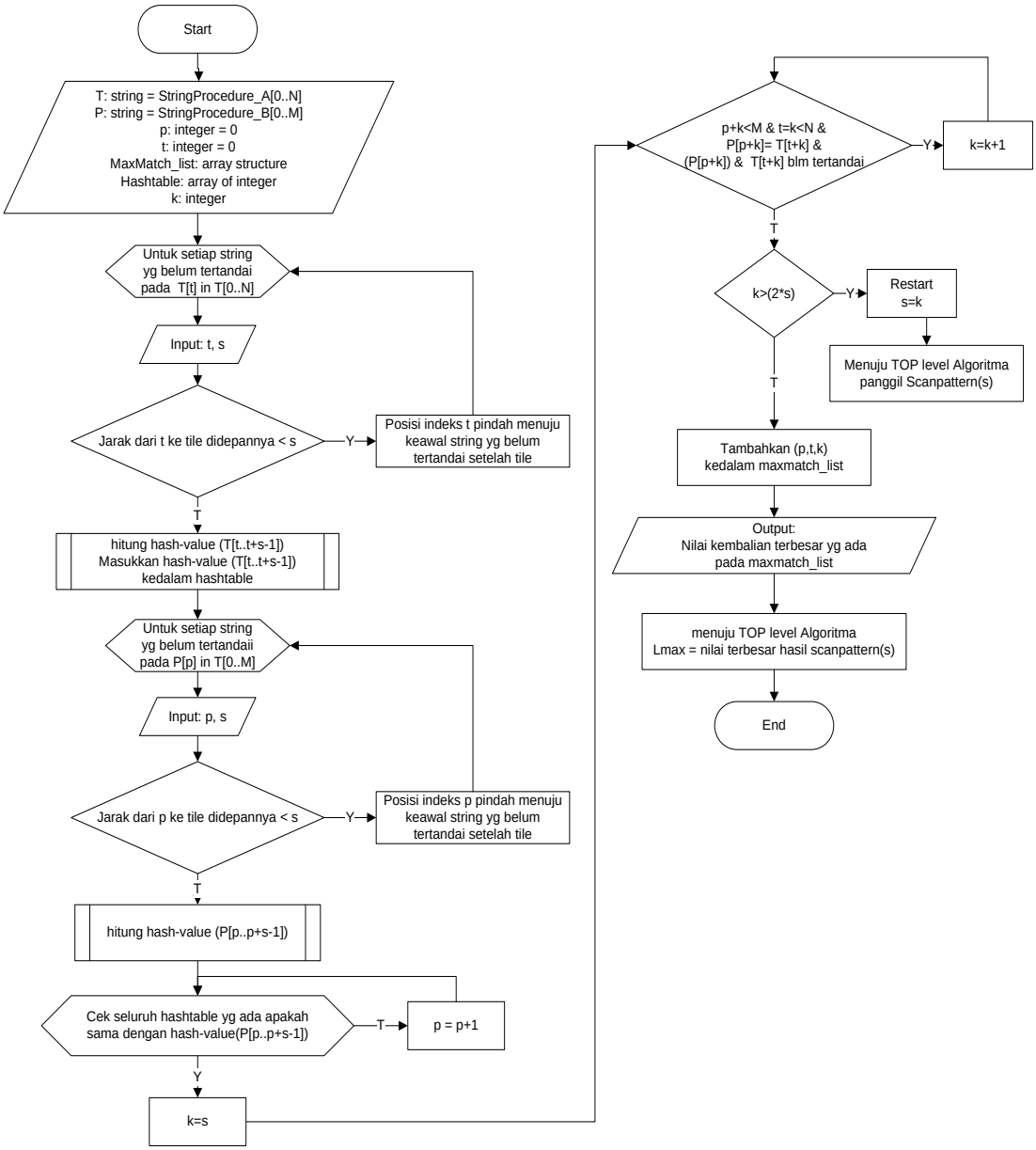
Telah dijelaskan sebelumnya pada sub bab 3.2.1, bahwa top level algoritma RKRGST menggambarkan langkah-langkah untuk menemukan string yang cocok sepanjang *search length (s)* yang dilakukan di fase *scanpattern*, kemudian akan dilakukan penandaan terhadap string yang memiliki kecocokan tersebut pada fase *marksing*. Flowchart dari top level algoritma RKRGST di gambarkan pada Gambar 4.6.



Gambar 4.6 Flowchart Top Level Algoritma RKRGST

4.3.5 Fase *scanpattern*

Fase *scanpattern* bertujuan mencari *substring* yang sama sepanjang *search length (s)*. Hasil pencarian *substring* yang sama disebut *maximal-match* dan akan disimpan kedalam *list-of-maximal-match*, nantinya akan dilakukan penandaan pada setiap karakter dari *maximal-match* tersebut di fase *markstring*. Keterangan tambahan mengenai fase *scanpattern* dapat dilihat pada pembahasan sebelumnya di sub bab 3.2.2.2. Gambar 4.7 merupakan Flowchart fase *scanpattern*.



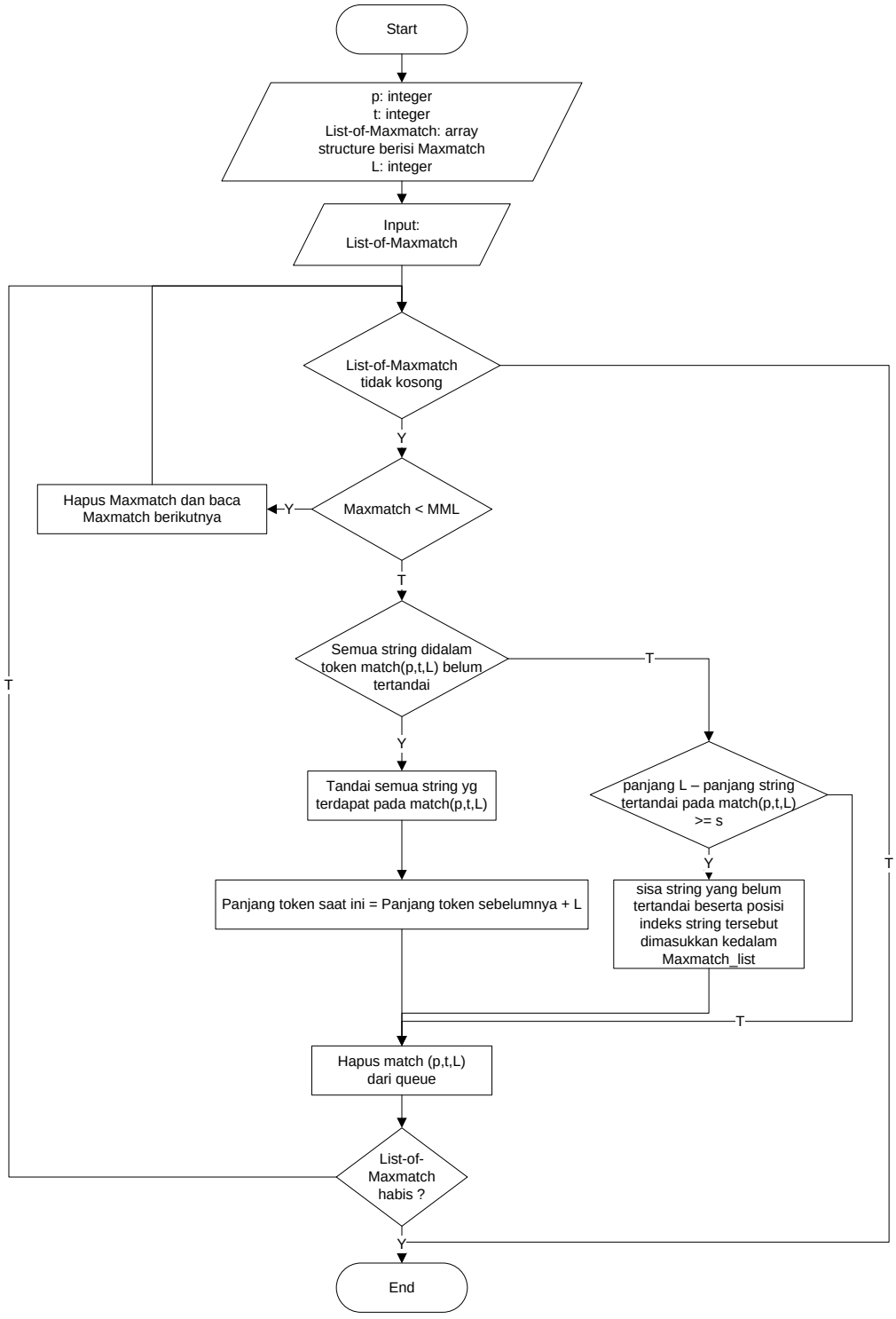
Gambar 4.7 Flowchart Fungsi *Scanpattern*

4.3.6 Fase *markstring*

Pada fase *markstring* akan dilakukan penandaan (*marked*) terhadap karakter-karakter yang terdapat didalam *list-of-maximal-match.* Berikut ini beberapa kondisi yang terjadi pada fase *markstring*.

1. Setiap *maximal-match* yang ada didalam *list-of-maximal-match* akan diuji apakah token-token yang menjadi bagian dari *maximal-match* tersebut telah tertandai (*marked*).
2. Jika telah tertandai, maka bagian dari *maximal-match* tersebut sebenarnya telah menjadi bagian dari *tile* yang lain, sehingga sebuah tile tidak dapat dibentuk lagi dari *maximal-match* tersebut.
3. Jika ada bagian dari maximal-match yang telah tertandai maka perlu dilakukan pengecekan apakah panjang karakter di *maximal-match* tersebut jika dikurangi jumlah string yang tertandai mencapai nilai yang sama atau lebih besar dari *s.* Jika kondisi ini terpenuhi maka sebuah *maximal-match* dapat dibentuk dari sisa karakter yang tidak tertandai pada *maximal-match* tersebut.
4. Jika semua token pada *maximal-match* belum ditandai, perlu dipastikan kembali apakah karakter-karakter yang ada didalamnya benar-benar memiliki kecocokan, jika kondisi ini terpenuhi maka sebuah *tile* dapat dibentuk dari *maximal-match* tersebut.

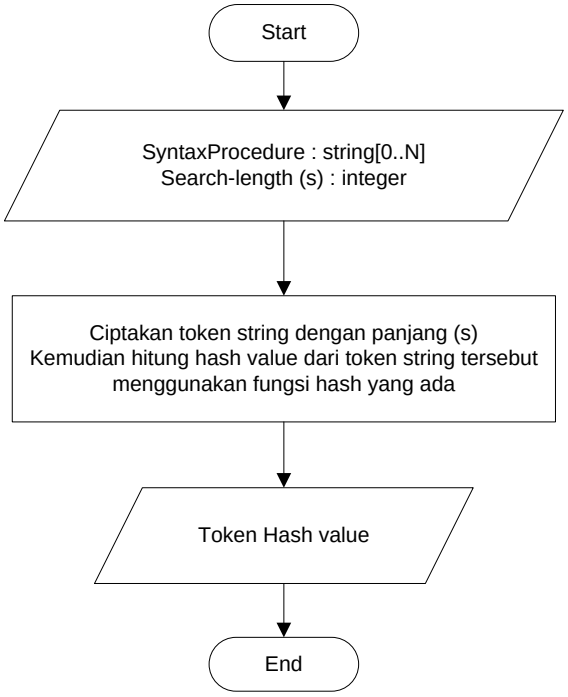
Pembahasan mengenai fase *markstring* sebelumnya telah disampaikan pada sub bab 3.2.2.3. Flowchart fase *markstring* digambarkan pada Gambar 4.8.



Gambar 4.8 Flowchart Fungsi Markstring

4.3.7 Fungsi *hash*

Fungsi *Hash* dilakukan untuk merubah *string* menjadi *token* yang berupa angka dengan panjang tertentu, dimana pada fase *scanpattern* algoritma RKRGST panjang *string* yang akan dibentuk *token* di inisialisasi oleh variabel *search length. Token* yang berupa angka tersebutlah yang akan dibandingkan dengan *token* lainnya yang terdapat pada *source code* pembandingnya. Mengenai rumus fungsi *hashing* telah disampaikan pada sub bab 3.3. Flowchart fungsi *hash* digambarkan pada Gambar 4.9.



Gambar 4.9 Flowchart Fungsi Hash

4.4 Proses *Scanning* dan *Parser* Pada Modul *Source Code*

Untuk Mendapatkan Syntax Yang Mewakili Isi Procedure :

1. Misalkan didalam suatu *source code* pemrograman visual basic terdapat suatu procedure seperti pada Gambar 4.10.

```
1   ⇨ Private Sub Form_Load()
2     Dim i as integer
3     'memanggil koneksi database
4     Call Connect
5     Frame1.Enabled = False
6         For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8         Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.10 Contoh procedure pada bahasa pemrograman visual basic

Simbol ⇨ menandakan proses *scanning* berada pada baris kode yang ditunjuk.

2. Untuk mengenali struktur procedure dari suatu bahasa pemrograman, pada contoh ini bahasa pemrograman visual basic, perlu dibangun sebuah *parser*. S*yntax* dasar yang akan digunakan untuk membangun *parser* agar dapat mengenali struktur procedure digambarkan pada Gambar 4.11

```
If Split(Trim(TextLine)," ")(0) = "Private" Or
   Split(Trim(TextLine)," ")(0) = "Function" Or
   Split(Trim(TextLine)," ")(0) = "Sub" Then
   ............
   ............
End If
```

Gambar 4.11 Syntax parser untuk mendeteksi procedure/function

Proses *scanning* dengan bantuan *parser* akan mampu mengenali suatu procedure berdasarkan delimeter procedure yang didefinisikan pada masing-masing bahasa pemrograman seperti pada Tabel 4.2.

Tabel 4.2 Pendefinisian Procedure/Function/Method pada beberapa bahasa pemrograman

Bahasa Pemrograman	**Syntax awal Procedure/Function/ Method**	**Syntax akhir Procedure/Function/ Method**
VB.6	Private, Function, Sub	End Sub
VB.Net	Private, Function, Sub	End Sub
C#	private, public, void	}
Delphi	procedure	end;

3. Pada proses *scanning* dibaris kode ke-1, *scanning* dengan bantuan *parser* mengenali adanya suatu pendifinisian procedure, maka yang dilakukan adalah menyimpan nama procedure tersebut kedalam stuktur data yang berupa array yang nantinya akan digunakan sebagai *filtering* terhadap pemanggilan *procedure* yang terdapat di dalam *body* suatu *procedure.* Kemudian dari proses pembacaan baris kode ke-1 ini, tidak ada *syntax* yang akan disimpan kedalam variable "syntax procedure". Sebagai catatan, variable "syntax procedure" adalah tempat menampungnya *syntax* yang akan mewakili isi *procedure* yang sedang di *scan*.
4. Selanjutnya proses *scanning* berpindah pada baris kode ke-2 seperti yang terlihat pada Gambar 4.12.

```
1     Private Sub Form_Load()
2  ⇨  Dim i as integer
3     'memanggil koneksi database
4     Call Connect
5     Frame1.Enabled = False
6         For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8         Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.12 Proses scanning baris ke-2

Pada proses *scanning* dibaris kode ke-2, dengan bantuan *parser* ditemukan adanya *reserved words* "Dim" yang merupakan pendifinisian dari sebuah variable yang ada pada bahasa pemrograman visual basic. Dengan ditemukannya reserved words "Dim", maka nama variable yang didefinisikan akan disimpan kedalam array, *syntax* dasar yang akan digunakan untuk menyimpan nama variable dapat dilihat pada Gambar 4.13.

```
1  For j = 0 To UBound(Split(CodeDibarisProcedure, " "))
2     CekSyntax = Split(CodeDibarisProcedure, " ")(j)
3     If CekSyntax = "Dim" Or CekSyntax = "Public" Then
4       ReDim Preserve ArrVariableModul1(n)
5       ArrVariableModul1(n) = Split(CodeDibarisProcedure, " ")(j+1)
6       n = n + 1
7     End If
8  Next
```

Gambar 4.13 Syntax parser untuk mendeteksi keberadaan variable

Nama variable yang telah disimpan nantinya akan digunakan untuk melakukan filter terhadap keberadaan suatu variable dalam baris kode berikutnya yang sedang di *scan.*

Pendifinisian variable secara eksplisit untuk bahasa pemrograman lainnya dapat dilihat pada Tabel 4.3.

Tabel 4.3 Pendefinisian variable pada beberapa bahasa pemrograman

Bahasa Pemrograman	**Cara Pendifinisikan Variable**
VB.6	**Dim** NamaVariabel **As** TipeData **Public** NamaVariabel **As** TipeData **Private** NamaVariabel **As** TipeData **Static** NamaVariabel **As** TipeData
VB.Net	**Dim** NamaVariabel **As** TipeData **Public** NamaVariabel **As** TipeData **Private** NamaVariabel **As** TipeData
C#	**TipeData** NamaVariabel Contoh : **byte** NamaVariabel **short** NamaVariabel **int** NamaVariabel **long** NamaVariabel **float** NamaVariabel **double** NamaVariabel **decimal** NamaVariabel **char** NamaVariabel **string** NamaVariabel **bool** NamaVariabel **object** NamaVariabel
Delphi	**Var** NamaVariabel **:** TipeData

5. Selanjutnya proses *scanning* berpindah pada baris kode ke-3 seperti yang terlihat pada Gambar 4.14.

```
1     Private Sub Form_Load()
2     Dim i as integer
3  ⇨  'memanggil koneksi database
4     Call Connect
5     Frame1.Enabled = False
6         For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8         Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.14 Proses scanning baris ke-3

Proses scanning pada baris kode ke-3, dengan bantuan parser ditemukan adanya *tanda baca* (‘) atau *single quote* yang menandakan sebuah komentar pada bahasa pemrograman visual basic. Dengan ditemukannya tanda baca *single quote*, maka baris kode tersebut akan langsung diabaikan dan proses scanning dilanjutkan pada baris kode berikutnya.

6. Selanjutnya proses *scanning* berada pada baris kode ke-4 seperti yang terlihat pada Gambar 4.15.

```
1     Private Sub Form_Load()
2     Dim i as integer
3     'memanggil koneksi database
4  ⇨  Call Connect
5     Frame1.Enabled = False
6         For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8         Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.15 Proses scanning baris ke-4

Proses *scanning* dibaris kode ke-4, dengan bantuan *parser* ditemukan adanya *reserved words “Call”* ini menandakan adanya pemanggilan terhadap suatu procedure, dengan demikian baris kode ini akan langsung diabaikan dan proses *scanning* berpindah menuju baris kode berikutnya.

7. Kemudian proses scanning berada pada baris kode ke-5 seperti yang terlihat pada Gambar 4.16.

```
1     Private Sub Form_Load()
2     Dim i as integer
3     'memanggil koneksi database
4     Call Connect
5  ⇨  Frame1.Enabled = False
6         For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8         Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.16 Proses scanning baris ke-5

Pada proses *scanning* dibaris kode ke-5, *parser* tidak menemukan adanya *reserved words*, dengan demikian *syntax* yang ada dibaris kde ke-5 akan ditampung kedalam variable “syntaxprocedure”, sehingga isi variable syntax procedure saat ini menjadi seperti pada Gambar 4.17.

```
SyntaxProcedure =  “Frame1.Enabled = False”
```

Gambar 4.17 Isi variable SyntaxProcedure

8. Selanjutnya pada baris kode ke-6 seperti yang terlihat pada Gambar 4.18

```
1     Private Sub Form_Load()
2     Dim i as integer
3     'memanggil koneksi database
4     Call Connect
5     Frame1.Enabled = False
6  ⇨      For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8         Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.18 Proses scanning baris ke-6

Saat membaca baris kode ke-6, parser menemukan adanya *reserved words* "For", akan tetapi reserved words "For" bukan merupakan *reserved words* untuk keperluan *parser*, karena *parser* yang dibangun adalah *parser* untuk keperluan mengenali struktur procedure, *parser* untuk mengenali pendifinisian variable, serta *parser* untuk mengenali pemanggilan suatu procedure yang ada didalam body suatu procedure, sehingga *syntax* pada baris kode ke-6 tidak terfilter oleh *parser* dengan demikian *syntax* dibaris kode ke-6 akan ditampung kedalam variable "SyntaxProcedure", sehingga isi variable *syntax* procedure saat ini menjadi seperti pada Gambar 4.19.

```
SyntaxProcedure =  "Frame1.Enabled = False For i = 2 to 5"
```

Gambar 4.19 Isi variable SyntaxProcedure

9. Selanjutnya pada baris kode ke-7 seperti yang terlihat pada Gambar 4.20.

```
1     Private Sub Form_Load()
2     Dim i as integer
3     'memanggil koneksi database
4     Call Connect
5     Frame1.Enabled = False
6         For i = 2 to 5
7  ⇨          Toolbar1.Buttons(i).Enabled = False
8         Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.20 Proses scanning baris ke-7

Pada proses *scanning* dibaris kode ke-7, *parser* tidak menemukan adanya *reserved words*, dengan demikian *syntax* yang ada dibaris kode ke-7 akan ditampung kedalam variable "SyntaxProcedure", sehingga isi variable "SyntaxProcedure" bertambah menjadi seperti pada Gambar 4.21.

```
SyntaxProcedure =  "Frame1.Enabled = False For i = 2 to 5
                    Toolbar1.Buttons(i).Enabled = False"
```

Gambar 4.21 Isi variable SyntaxProcedure

10. Pada baris kode ke-8, seperti yang terlihat pada Gambar 4.22.

```
1     Private Sub Form_Load()
2     Dim i as integer
3     'memanggil koneksi database
4     Call Connect
5     Frame1.Enabled = False
6         For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8  ⇨      Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.22 Proses scanning baris ke-8

proses *scanning* dibaris kode ke-7, terdapat *reserved words* "Next" akan tetapi *reserved words* "Next" tersebut tidak merupakan bagian dari *reserved words* yang digunakan untuk keperluan *parser*, sehingga baris kode ke-8 akan ditampung kedalam variable "SyntaxProcedure", sehingga isi variable syntax procedure saat ini menjadi seperti pada Gambar 4.23.

```
SyntaxProcedure =  "Frame1.Enabled = False For i = 2 to 5
                    Toolbar1.Buttons(i).Enabled = False Next"
```

Gambar 4.23 Isi variable SyntaxProcedure

11. Selanjutnya pada baris kode ke-9 seperti yang terlihat pada Gambar 4.24

```
1     Private Sub Form_Load()
2     Dim i as integer
3     'memanggil koneksi database
4     Call Connect
5     Frame1.Enabled = False
6         For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8         Next
9  ⇨  Call ShowDataGrid
10    End Sub
```

Gambar 4.24 Proses scanning baris ke-9

Proses *scanning* dibaris kode ke-9, dengan bantuan *parser* ditemukan adanya *reserved words "Call",* ini menandakan adanya pemanggilan terhadap suatu procedure. Dengan demikian baris kode ini akan langsung diabaikan dan proses *scanning* berpindah menuju baris kode berikutnya.

12. Selanjutnya pada baris kode ke-10. Proses *scanning* dengan bantuan *parser* menemukan adanya *reserved words* "End Sub" seperti yang terlihat pada Gambar 4.25, yang menandakan bahwa ini adalah akhir dari suatu procedure pada bahasa pemrograman visual basic. Untuk bahasa pemrograman lain dapat dilihat pada table 4.2 pada pembahasan sebelumnya di point 2.

```
1     Private Sub Form_Load()
2     Dim i as integer
3     'memanggil koneksi database
4     Call Connect
5     Frame1.Enabled = False
6         For i = 2 to 5
7             Toolbar1.Buttons(i).Enabled = False
8         Next
9     Call ShowDataGrid
10    End Sub
```

Gambar 4.25 Proses scanning baris ke-10

13. Langkah selanjutnya setelah akhir dari procedure dicapai, selanjutnya melakukan filtering variable, filtering stopword, filtering tandabaca dan case folding terhadap isi dari variable "SyntaxProcedure".
14. Isi variable "SyntaxProcedure" sebelum dilakukan filtering variable dapat dilihat seperti pada Gambar 4.26.

```
SyntaxProcedure = "Frame1.Enabled = False For i = 2 to 5
                  Toolbar1.Buttons(i).Enabled = False Next"
```

Gambar 4.26 Isi variable SyntaxProcedure

Kemudian dilakukan filtering terhadap keberadaan variable yang telah didapat pada proses di point 4, sehingga isi variable "SyntaxProcedure" menjadi seperti pada Gambar 4.27.

```
SyntaxProcedure = "Frame1.Enabled = False For = 2 to 5
                  Toolbar1.Buttons().Enabled = False Next"
```

Gambar 4.27 Isi variable SyntaxProcedure setelah filtering variable

15. Selanjutnya dilakukan filtering Stopword yaitu menghapus keyword-keyword tertentu yang tidak ingin digunakan pada proses pembentukan token procedure. *Keyword* tersebut bisa saja berupa *reserved words* misalnya "For, Next, If, Then, Else" dan lain sebagainya. Sehingga pada tahap filtering stopword ini, isi dari variable "SyntaxProcedure" menjadi seperti pada Gambar 4.28.

```
SyntaxProcedure =  "Frame1.Enabled = False = 2 to 5
                    Toolbar1.Buttons().Enabled = False"
```

Gambar 4.28 Isi variable SyntaxProcedure setelah filtering stopword

16. Langkah berikutnya filtering tanda-baca, yaitu menggantikan seluruh tanda baca yang ada didalam isi dari variable "SyntaxProcedure" menjadi *white space*, sehingga pada tahap filtering tanda-baca ini, isi dari variable "SyntaxProcedure" menjadi seperti pada Gambar 4.29.

```
SyntaxProcedure =  “Frame1 Enabled  False  2 to 5 Toolbar1 Buttons
                    Enabled  False”
```

Gambar 4.29 Isi variable SyntaxProcedure setelah filtering tanda-baca

17. Proses selanjutnya merubah seluruh huruf yang ada di variable “SyntaxProcedure” menjadi huruf kecil dan menghilangkan karakter spasi yang ada, sehingga isi dari variable “SyntaxProcedure” menjadi seperti pada Gambar 4.30.

```
SyntaxProcedure =  “frame1enabledfalse2to5toolbar1buttonsenabledfalse”
```

Gambar 4.30 Casefolding isi variable SyntaxProcedure

18. Setelah melalui beberapa tahapan diatas, *string* yang terdapat pada variable “SyntaxProcedure” telah mewakili isi dari *procedure* yang sedang diproses, dan untuk *procedure* lain yang terdapat dalam *source code* uji maupun *source code* pembandingnya, akan dilakukan hal yang sama seperti pada langkah-langkah sebelumnya.
19. Setelah semua *string* yang mewakili isi *procedure* yang terdapat pada kedua *source code* didapat, kemudian untuk setiap *string* yang mewakili *procedure* tersebut akan dicek kemiripannya menggunakan algoritma RKRGST. Mengenai Proses pengecekan kemiripan *string* menggunakan algoritma RKRGST dapat dilihat pada halaman lampiran.
20. Setelah didapat jumlah pasangan procedure yang sama diantara kedua modul *source code*, persentase kemiripan kedua *source code* dihitung menggunakan metode dice coefficient dengan rumus ((*n**2)/(t1+t2)*100), dimana *n* merupakan jumlah procedure hasil pembandingan yang dianggap sama oleh sistem dan t1,t2 merupakan jumlah masing-masing procedure yang terdapat pada *source code* pengujian. Suatu procedure dianggap memiliki kesamaan jika jumlah string yang tertandai pada procedure yang diuji mencapai lebih besar atau sama dengan nilai sensitifitas deteksi yang telah ditentukan pada saat pengujian.

4.5 Perancangan HIPO (Hierarchy plus Input-Process-Output)

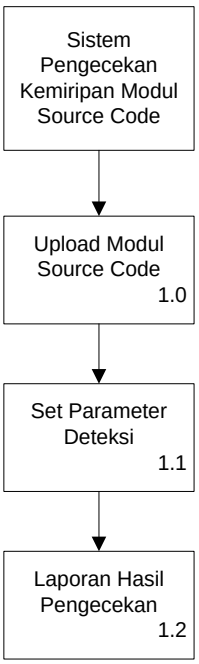
HIPO merupakan alat bantu untuk merancang dan mendokumentasikan siklus sistem yang dikembangkan dan didukung oleh perusahaan IBM. HIPO

menggambarkan suatu struktur bertingkat guna memahami fungsi-fungsi dari modul-modul suatu sistem, dan HIPO juga dirancang untuk menggambarkan modul-modul yang harus diselesaikan oleh programmer. HIPO tidak dipakai untuk menunjukkan instruksi-instruksi program yang akan digunakan. HIPO menyediakan penjelasan yang lengkap dari input yang akan digunakan, proses yang akan dilakukan serta output yang diinginkan.

Fungsi-fungsi dari sistem digambarkan oleh HIPO dalam tiga tingkatan dan untuk masing-masing tingkatan digambarkan dalam bentuk diagram tersendiri seperti berikut :

a. Visual Table of Contents

Gambar 4.31 merupakan diagram yang menggambarkan hubungan dari modul-modul dalam suatu sistem secara berjenjang



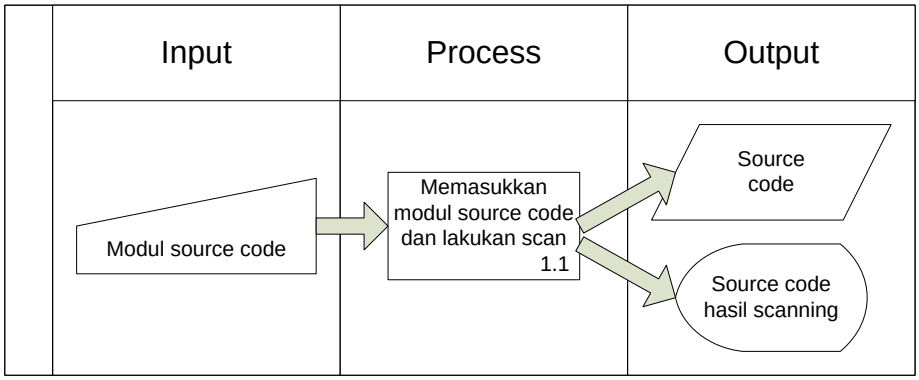
Gambar 4.31 Visual table of contents dari sistem pengecekan kemiripan modul source code

b. Overview Diagram

Overview diagram digunakan untuk menunjukkan secara garis besar hubungan dari input, proses dan output, dimana bagian input menunjukkan item-

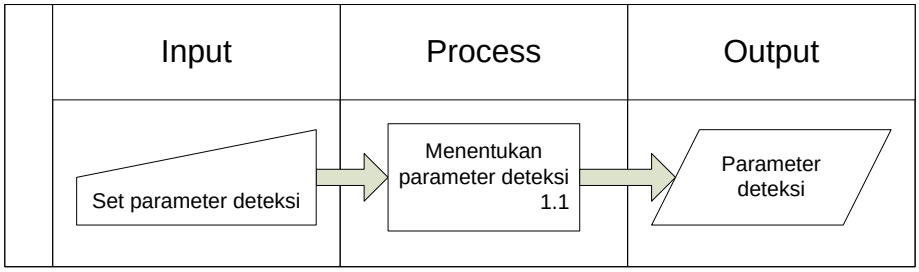
item data yang akan digunakan oleh bagian proses, sedangkan bagian proses berisi langkah-langkah yang menggambarkan kerja dari fungsi atau modul serta bagian output berisi hasil pemrosesan data.

Overview diagram pada modul 1.0 untuk upload modul source code digambarkan pada Gambar 4.32



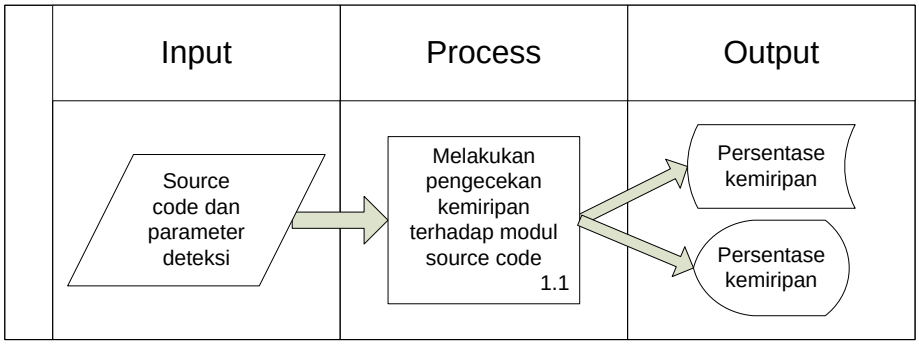
Gambar 4.32 Overview Diagram Modul 1.0

Overview diagram pada modul 1.1 untuk melakukan set parameter deteksi digambarkan pada Gambar 4.33



Gambar 4.33 Overview Diagram Modul 1.1

Overview diagram pada modul 1.2 untuk menghasilkan laporan hasil deteksi sistem digambarkan pada Gambar 4.34



Gambar 4.34 Overview Diagram Modul 1.2

c. Detail Diagram

Detail Diagram berisi elemen-elemen dasar dari paket yang menggambarkan secara rinci cara kerja dari modul 1.0, modul 1.1 dan modul 1.2, masing-masing detail diagram dapat dijelaskan pada Tabel 4.4, 4.5 dan 4.6.

Tabel 4.4 Detail Diagram Modul 1.0

Input	Process	Output
Modul Source Code : Terdiri dari serangkaian procedure/function/method yang didalamnya terdapat beberapa statement, variable, procedure calling dan lain sebagainya	User memasukkan modul *source code* dan sistem melakukan *scanning source code* baris perbaris kemudian menampilkannya kedalam sebuah TextBox	Source code hasil *scanning*

Tabel 4.5 Detail Diagram Modul 1.1

Input	Process	Output
Parameter Deteksi Meliputi : - Filtering variable - Filtering Comment - Disable Procedure Call - Ignore name of procedure - Sensitifitas deteksi	Menentukan parameter deteksi	Parameter deteksi

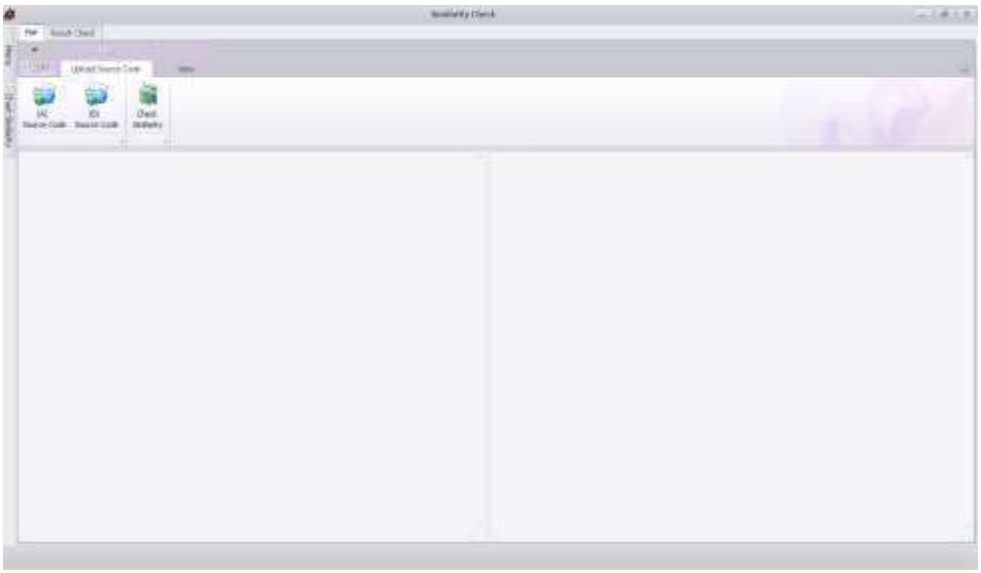
Tabel 4.6 Detail Diagram Modul 1.2

Input	Process	Output
Source code dan parameter deteksi yang telah ditentukan oleh user	Melakukan pengecekan kemiripan pada modul source code berdasarkan parameter deteksi yang telah ditentukan oleh user	Persentase kemiripan

4.6 Perancangan Antar Muka

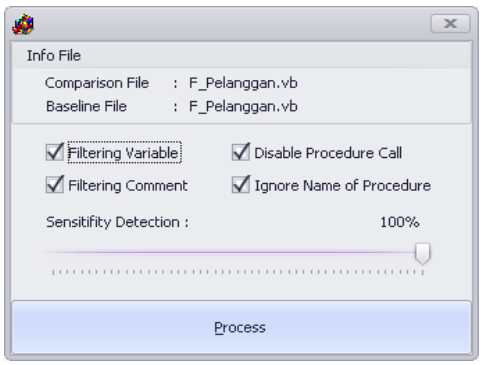
Bagian terakhir dari rancangan sistem ini adalah desain antarmuka (*interface*) dari aplikasi.

Pada perancangan antarmuka proses upload modul, pengguna diminta memasukkan data modul dengan menekan tombol ‘Source Code’, kemudian akan dilakukan *scanning* oleh sistem untuk membaca *syntax-syntax* yang ada pada modul yang kemudian akan ditampung pada textbox1 dan textbox2 seperti yang terlihat pada Gambar 4.35.



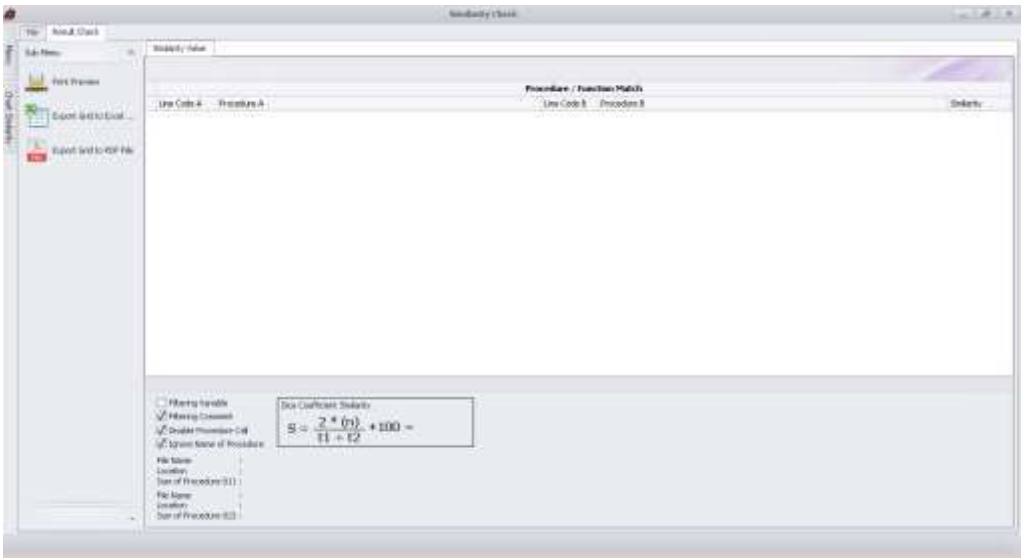
Gambar 4.35 Rancangan antar muka proses upload modul

Pada rancangan antarmuka set parameter proses, sebelum memulai melakukan proses, pengguna diminta menentukan parameter proses seperti yang terlihat pada Gambar 4.36.



Gambar 4.36 Rancangan antar muka set parameter proses

Perancangan antarmuka informasi hasil proses dapat dilihat pada Gambar 4.37. Pada antar muka ini akan ditampilkan informasi mengenai jumlah kemiripan modul token, persentase kemiripan, nomor baris kode program dan nama-nama prosedure yang memiliki kemiripan.



Gambar 4.37 Rancangan antar muka informasi hasil proses

BAB V
IMPLEMENTASI

Spesifikasi hardware dan software yang digunakan untuk merancang, membuat dan menjalankan sistem ini menggunakan processor intel dual core 2.0 GHz, Memori 2Gb, dengan sistem operasi Windows XP Professional 32 bit, dan menggunakan bahasa pemrograman Visual Basic .Net (VB.Net) dengan Tools Integrated Development Environment yang digunakan adalah Visual Studio 2010, serta menggunakan komponent tambahan milik DevExpress.

5.1 Implementasi Upload Modul *Source Code*

Gambar 5.1 merupakan *code snippets* yang digunakan dalam proses upload modul *source code*, proses upload modul dilakukan oleh user pengguna aplikasi, pada baris kode ke-15 sistem akan memanggil procedure ReadSourceCode dengan menyertakan 3 parameter (lokasi modul, kode upload dan ekstensi program) yang dikirim untuk proses *scanning* terhadap modul yang diupload.

```
1   Private Sub BarButtonItem1_ItemClick(ByVal sender As Sistem.Object, ByVal e As
    DevExpress.XtraBars.ItemClickEventArgs) Handles BarButtonItem1.ItemClick
2       Dim CodeTextbox As Byte = 1
3       Dim OpenFileDialog As New OpenFileDialog
4
5       OpenFileDialog.InitialDirectory =
        My.Computer.FileSistem.SpecialDirectories.MyDocuments
6       OpenFileDialog.ShowDialog(Me)
7
8       If OpenFileDialog.FileName <> "" Then
9          SourceCodeA.Text = ""
10         Dim xPath As String = OpenFileDialog.FileName
11         NamaModulA.Text = OpenFileDialog.SafeFileName
12         LokasiModulA.Text = xPath
13         Dim xExtention As String =
14         Split(OpenFileDialog.SafeFileName, ".")(1)
15         ReadSourceCode(xPath, CodeTextbox, xExtention)
16      End If
17  End Sub
```

Gambar 5.1 Code snippets proses upload modul

5.2 Implementasi Mengambil Data *Stopword*

Untuk mengambil data *stopword* diperlukan ekstensi dari modul *source code* yang diupload user pengguna aplikasi. Variable "extensi" merupakan parameter yang dikirim dari pemanggilan procedure "FilteringStopWord" yang berisi ekstensi dari modul *source code*.

Data *stopword* yang disimpan pada file bertipe data .txt akan diambil berdasarkan isi dari variable ekstensi tersebut, selanjutnya akan disimpan kedalam sebuah array yang nantinya akan dipakai guna keperluan filtering *stopword*. *Code Snippets* untuk mengambil data stopword digambarkan pada Gambar 5.2.

```
1   Private Sub FilteringStopWord(ByVal extensi As String)
2   Dim objReader As New System.IO.StreamReader("D:\copascheck\filter.txt")
3   Dim vExtensi As String
4   Dim vStopWord As String
5   Dim vTextLine As String
6   Do While objReader.Peek() <> -1
7       vTextLine = objReader.ReadLine()
8       vExtensi = Split(vTextLine, ":")(0)
9       vStopWord = Split(vTextLine, ":")(1)
10      If vExtensi = extensi Then
11         ReDim Preserve ArrFilter(UBound(Split(vStopWord, ",")))
12         For i = 0 To UBound(Split(vStopWord, ","))
13             ArrFilter(i) = Split(vStopWord, ",")(i)
14         Next
15         Exit Do
16      End If
17  Loop
18  End Sub
```

Gambar 5.2 Code snippets Mengambil Data Stopword

5.3 Implementasi Scanning dan Parser pada Modul Source Code

Proses scanning modul dilakukan oleh function *ReadSourceCode* menggunakan class Stream Reader dari library I/O.

5.3.1 *Scanning* dan *parser source code* VB 6.0

Implementasi *scanning* dan *parser* pada *source code* VB 6.0 digambarkan pada Gambar 5.3. Baris kode ke-13 merupakan kondisi pengecekan yang menyatakan apakah variable xExtention berisi "frm" dimana "frm" merupakan ekstensi dari bahasa pemrograman VB.6.0. Proses pembacaan *source code* dilakukan secara *line by line* yang ditunjukkan pada baris kode ke-15. Pada saat melakukan ekstraksi modul pemrograman vb.6.0 akan ditemukan hasil ekstraksi berupa meta data dari modul yang diupload, meta data tersebut tidak digunakan untuk proses pengecekan sehingga diperlukan proses penyaringan hasil ekstraksi

untuk mengambil data hasil ekstraksi yang diinginkan, untuk itu pada baris kode ke-17 menjadi parameter sistem untuk penyaringan data, dimulai dengan membaca sebuah procedure, function, variable dimana pada bahasa pemrograman VB.6.0 *syntax* dalam pembuatan procedure diawali dengan keyword “Private” dan ”Sub” sedangkan sebuah function diawali dengan keyword “Function” dan sebuah variable diawali dengan keyword “Dim”. Setelalah kondisi baris kode ke-17 terpenuhi maka sistem akan mencatat *syntax* hasil *scanning* kedalam sebuah *textbox* yang nantinya *syntax-syntax* yang ada pada *textbox* tersebut yang akan dibandingkan satu sama lain pada procedure yang ada didalam modul uji.

Pada baris kode ke-22, dimana saat proses scanning menemukan sebuah procedure atau function maka nama dari procedure atau function tersebut beserta no.baris kode-nya akan ditampung kedalam sebuah struktur data yang berupa *array structure*. Kumpulan array yang berisi nama-nama procedure atau function tersebut akan dijadikan parameter untuk menyaring proses pemanggilan procedure yang mungkin saja akan ada pada procedure lainnya.

Penjelasan diatas juga diterapkan untuk bahasa pemrograman lainnya, hanya saja parameter pembacaan procedure atau function dan variable disesuaikan dengan bahasa pemrograman yang menjadi target deteksi.

```
1   Function ReadSourceCode(ByVal xFilePath As String, ByVal xCode As String,
2   ByVal xExtention As String)
3   Dim TextLine As String
4   Dim i As Short = 1
5   Dim vNo As Short
6   Dim xCatat As Boolean
7
8   If Sistem.IO.File.Exists(xFilePath) = True Then
9         Dim objReader As New Sistem.IO.StreamReader(xFilePath)
10        Dim LineNumber As String
11        Cursor = Cursors.WaitCursor
12        '+++++++++++++++++ Read Source Code VB.6 ++++++++++++++++++++++++++
13        If xExtention = "frm" Then
14        Do While objReader.Peek() <> -1
15           TextLine = objReader.ReadLine() & vbNewLine
16           TextLine = Trim(TextLine)
17           If Split(Trim(TextLine), " ")(0) = "Private"
                Or Split(Trim(TextLine), " ")(0) = "Function" Or
                Split(Trim(TextLine), " ")(0) = "Sub" Or
                Split(Trim(TextLine), " ")(0) = "Dim" Then
18              LineNumber = i
19              If xCode = 1 Then
20                 SourceCodeA.Text = SourceCodeA.Text & (LineNumber.PadRight(5,
                   "*")) & "|| " & TextLine
21                 If TextLine <> "" Then
22                    If Split(Trim(TextLine), " ")(0) = "Private" Or
                         Split(Trim(TextLine), " ")(0) = "Function" Or
```

```
                        Split(Trim(TextLine), " ")(0) = "Sub" Then
23                      ReDim Preserve ArrDataModulA(vNo)
24                      ArrDataModulA(vNo).LineCode = i
25                      ArrDataModulA(vNo).ProcedureName = Trim(TextLine)
26                      vNo = vNo + 1
27                      SumProcedureT1.Text = UBound(ArrDataModulA) + 1
28                   End If
29                End If
30             Else
31                SourceCodeB.Text = SourceCodeB.Text & (LineNumber.PadRight(5,
                  "*")) & "|| " & TextLine
32                If TextLine <> "" Then
33                   If Split(Trim(TextLine), " ")(0) = "Private" Or
                        Split(Trim(TextLine), " ")(0) = "Function" Or
                        Split(Trim(TextLine), " ")(0) = "Sub" Then
34                      ReDim Preserve ArrDataModulB(vNo)
35                      ArrDataModulB(vNo).LineCode = i
36                      ArrDataModulB(vNo).ProcedureName = Trim(TextLine)
37                      vNo = vNo + 1
38                      SumProcedureT2.Text = UBound(ArrDataModulB) + 1
39                   End If
40                End If
41             End If
42             i = i + 1
43             xCatat = True
44
45          ElseIf xCatat = True Then
46             If xCode = 1 Then
47                LineNumber = i
48                SourceCodeA.Text = SourceCodeA.Text & (LineNumber.PadRight(5,
                  "*")) & "|| " & TextLine
49             Else
50                LineNumber = i
51                SourceCodeB.Text = SourceCodeB.Text & (LineNumber.PadRight(5,
                  "*")) & "|| " & TextLine
52             End If
53             i = i + 1
54          End If
55       Loop
..    ……..
..    ……..
219   End Function
```

Gambar 5.3 Code Snippets Scanning dan Parser Source Code VB 6.0

5.3.2 *Scanning* dan *parser* dalam mengenali struktur *procedure* VB 6.0

Implementasi *scanning* dan *parser* dalam mengenali struktur procedure pada bahasa pemrograman VB 6.0 digambarkan pada Gambar 5.4. Baris kode ke-14 merupakan array bertipe data string yang digunakan untuk menampung isi dari kode program yang terdapat pada textbox SourceCodeB.Text yang merupakan hasil dari proses scanning dan parser pada sub bab 5.3.1.

Variable *vStringLine* yang terdapat pada baris kode ke 17-18 merupakan variable yang menampung baris kode yang sedang di-*scan* dengan terlebih dahulu menghilangkan angka yang terdapat diawal baris kode tersebut. Dengan bantuan

fungsi *split* dengan karakter pemisah "||", variable *vStringLine* inilah yang kemudian akan dicek apakah syntax awal yang terdapat didalam isi dari variable *vStringLine* tersebut mengandung *string delimeter procedure.*

Jika string *delimeter procedure* untuk pendefinisian procedure terdapat diawal syntax *vStringLine*, maka isi dari variable *vStringLine* tersebut akan disimpan kedalam variable *sString* seperti pada baris kode ke-62 dan baris kode lainnya yang terdapat variable *sString*.

Pengecekan yang dilakukan pada baris kode ke-42, 43 serta baris kode 74 dan 75 adalah untuk memeriksa apakah syntax awal dari isi variable *vStringLine* mengandung string *delimeter procedure* yang menyatakan akhir dari suatu procedure pada bahasa pemrograman visual basic.

Jika kondisi itu terpenuhi maka akan dilakukan proses filtering variable, filtering *stopword*, filtering tanda baca serta *casefolding* pada isi variable *sString*, selanjutnya isi variable sString disimpan kedalam sebuah array structure yang nantinya akan digunakan ditahap selanjutnya.

```
1   Private Sub CekModul_VB_Baseline()
2   Dim vStringLine As String
3   Dim vIndexProcedure As Byte = 0
4   Dim vCekVariable As String
5   Dim vMulai As Integer
6   Dim sString As String
7   Dim lines() As String
8   Dim i As Integer
9   Dim j As Integer
10  Dim k As Integer
11  Dim l As Integer
12  Dim n As Integer
13
14  Erase lines
15  lines = Split(SourceCodeB.Text, vbCrLf)
16      For i = 0 To (UBound(lines) - 1)
17              vStringLine = Split(lines(i), "||")(1)
18              vStringLine = Trim(vStringLine)
19              'jika membaca sebuah procedure maka abaikan
20              If Trim(Split(vStringLine, " ")(0)) = "Private" Or
                   Trim(Split(vStringLine, " ")(0)) = "Function" Or
                   Trim(Split(vStringLine, " ")(0)) = "Sub" Then
21                 If vStringLine <> "" Then
22                    vMulai = 1
23                 End If
24              ElseIf Mid(Split(vStringLine, " ")(0), 1, 1) = "'" Then
25                  'Abaikan Comment
26              ElseIf vStringLine = "" Or vStringLine = "  " Then
27                  'abaikan string kosong
28              Else
29                  If vSetFilterVariable = 1 Then
30                  'Request proses menggunakan filtering variable
31                  'Jika mengandung variable maka Simpan variable tersebut
```

```
32                  'untuk kemudian digunakan sebagai filter
33                  '===Cek Keberadaan variable
34                      (... ... ... ... ...)
35                      (... ... ... ... ...)
36                     '====Jika bukan Inisialisasi variable dan
37                     'bukan pemanggilan sebuah procedure
38                     'maka add vStringLine kedalam sString
39                     If Trim(Split(vStringLine, " ")(0)) <> "Dim" And
                          Trim(Split(vStringLine, " ")(0)) <> "Call" Then
40                        'cek apakah akhir sebuah procedure jika "ya" maka
41                        'keluar jika "tidak" maka catat di textbox
42                        If Trim(Split(vStringLine, " ")(0)) = "End" Then
43                           If Split(Trim(vStringLine), " ")(1) = "Sub" Then
44                              '===Filtering variable
45                               (... ... ... ... ...)
46                              '===Case folding
47                               (... ... ... ... ...)
48                              '===Filtering StopWord
49                               (... ... ... ... ...)
50                              'Replace tanda baca
51                               (... ... ... ... ...)
52                           ArrDataModulB(vIndexProcedure).TextProcedure = sString
53                           vIndexProcedure = vIndexProcedure + 1
54                           'clear
55                           sString = ""
56                                 Else
57                                     vStringLine = Split(lines(i), "| ")(1)
58                                     sString = Trim(sString) & " " & (vStringLine)
59                                 End If
60                             Else
61                                 vStringLine = Split(lines(i), "| ")(1)
62                                 sString = Trim(sString) & " " & Trim(vStringLine)
63                             End If
64
65                      End If
66                  Else
67                      'Jika variable tidak di Filter
68                      '====Jika bukan Inisialisasi variable dan
69                      'bukan pemanggilan sebuah procedure
70                      '====maka add vStringLine kedalam sString
71                      If Trim(Split(vStringLine, " ")(0)) <> "Dim" And
                           Trim(Split(vStringLine, " ")(0)) <> "Call" And
                           vMulai <> 0 Then
72                         'cek apakah akhir sebuah procedure jika "ya" maka
73                         'keluar jika "tidak" maka catat di textbox
74                         If Trim(Split(vStringLine, " ")(0)) = "End" Then
75                            If Trim(Split(vStringLine, " ")(1)) = "Sub" Then
76                              '===Case folding
77                               (... ... ... ... ...)
78                              '===Filtering StopWord
79                               (... ... ... ... ...)
80                              'Replace tanda baca
81                               (... ... ... ... ...)
82                               ArrDataModulB(vIndexProcedure).TextProcedure =
83                               sString
84                               vIndexProcedure = vIndexProcedure + 1
85                               'clear
86                               sString = ""
87                             Else
88                               vStringLine = Split(lines(i), "| ")(1)
89                               sString = Trim(sString) & " " &
90                                         Trim(vStringLine)
91                            End If
92                         Else
```

```
93                              vStringLine = Split(lines(i), "| ")(1)
94                              sString = Trim(sString) & " " & Trim(vStringLine)
95                          End If
96                      ElseIf Trim(Split(vStringLine, " ")(0)) <> "Call" And
97                              vMulai <> 0 Then
98                              vStringLine = Split(lines(i), "| ")(1)
99                              sString = Trim(sString) & " " & Trim(vStringLine)
100                     End If
101                 End If
102             End If
103         Next
    End Sub
```

Gambar 5.4 Code Snippets Scanning dan Parser Struktur Procedure VB 6.0

5.3.3 *Scanning* dan *Parser variable* pada *source code* VB 6.0

Proses scanning dan parser variable merupakan bagian dari *code snippets* pada Gambar 5.4 diatas, yang terletak pada baris kode ke 34 dan 35. Proses ini merupakan proses menganalisa keberadaan variable yang terdapat pada baris kode yang saat itu sedang di *scan.*

Dengan cara memisahkan kode program berdasarkan karakter *whitespace* (“ “), dan melakukan perulangan menggunakan “*for*”, kemudian melakukan pengecekan satu persatu apakah variable vCekVariable=”Dim” atau vCekVariable=”Public” dimana Dim dan Public adalah string delimeter untuk pendifinisian variable pada bahasa pemrograman vb.6.0.

Jika kondisi diatas terpenuhi, maka nama variable akan ditemukan dengan bantuan kode `Split(vStringLine, " ")(j + 1)` seperti yang dijelaskan pada baris kode ke-5 pada Gambar 5.7, kemudian nama variable akan disimpan kedalam suatu array untuk keperluan filtering keberadaan variable pada text procedure. Proses *scanning* dan *parser* variable selengkapnya digambarkan pada Gambar 5.5.

```
1   For j = 0 To UBound(Split(vStringLine, " "))
2       vCekVariable = Split(vStringLine, " ")(j)
3       If vCekVariable = "Dim" Or vCekVariable = "Public" Then
4          ReDim Preserve ArrVariableModul1(n)
5          ArrVariableModul1(n) = Split(vStringLine, " ")(j + 1)
6          n = n + 1
7       End If
8   Next
```

Gambar 5.5 Code Snippets Scanning dan Parser Variable VB 6.0

5.3.4 Scanning dan Parser Source Code VB.Net

Implementasi scanning dan parser pada *source code* VB.Net digambarkan pada Gambar 5.6. Baris kode ke-58 merupakan kondisi pengecekan yang menyatakan apakah variable xExtention berisi “vb”, dimana “vb” merupakan

ekstensi dari bahasa pemrograman VB.Net. Proses pembacaan *source code* dilakukan secara *line by line* yang ditunjukkan pada baris kode ke-60, baris kode ke-62 menjadi parameter sistem untuk merekam data, diawali dengan *keyword* "*Public*" yang menyatakan suatu *Class* pada modul, atau diawali dengan *keyword "Import"* yang biasa digunakan untuk menambahkan *library* yang akan dipakai.

Pada baris kode ke-76, dimana saat proses *scanning* menemukan sebuah *procedure* atau *function* maka nama dari *procedure* atau *function* tersebut beserta no.baris kode-nya akan ditampung kedalam sebuah struktur data yang berupa *array structure*, dimana kumpulan array yang berisi nama-nama *procedure* atau *function* tadi akan dijadikan parameter untuk menyaring proses pemanggilan *procedure* yang mungkin saja akan ada pada *procedure-procedure* lainnya.

```
1   Function ReadSourceCode(ByVal xFilePath As String, ByVal xCode As String,
2   ByVal xExtention As String)
3   Dim TextLine As String
4   Dim i As Short = 1
5   Dim vNo As Short
6   Dim xCatat As Boolean
7
8   If Sistem.IO.File.Exists(xFilePath) = True Then
9         Dim objReader As New Sistem.IO.StreamReader(xFilePath)
10        Dim LineNumber As String
11        Cursor = Cursors.WaitCursor
..    ..............
..    ..............
      '++++++++++++++++++++ Read Source Code VB.Net ++++++++++++++++++++++++++
58    ElseIf xExtention = "vb" Then
59       Do While objReader.Peek() <> -1
60          TextLine = objReader.ReadLine() & vbNewLine
61          TextLine = Trim(TextLine)
62          If Split(Trim(TextLine), " ")(0) = "Public" Or
               Split(Trim(TextLine),"")(0) = "Imports" Then
63             LineNumber = i
64             If xCode = 1 Then
65                SourceCodeA.Text = SourceCodeA.Text & (LineNumber.PadRight(5,
                  "*")) & "|| " & TextLine
66             Else
67                SourceCodeB.Text = SourceCodeB.Text & (LineNumber.PadRight(5,
                  "*")) & "|| " & TextLine
68             End If
69             i = i + 1
70             xCatat = True
71          ElseIf xCatat = True Then
72             If xCode = 1 Then
73                LineNumber = i
74                SourceCodeA.Text = SourceCodeA.Text & (LineNumber.PadRight(5,
                  "*")) & "|| " & TextLine
75                If TextLine <> "" Then
76                   If Split(Trim(TextLine), " ")(0) = "Private" Or
                        Split(Trim(TextLine), " ")(0) = "Function" Or
                        Split(Trim(TextLine), " ")(0) = "Sub" Then
77                      ReDim Preserve ArrDataModulA(vNo)
78                      ArrDataModulA(vNo).LineCode = i
```

```
79                      ArrDataModulA(vNo).ProcedureName = Trim(TextLine)
80                      vNo = vNo + 1
81                      SumProcedureT1.Text = UBound(ArrDataModulA) + 1
82                   End If
83                End If
84             Else
85                LineNumber = i
86                SourceCodeB.Text = SourceCodeB.Text & (LineNumber.PadRight(5,
                  "*")) & "|| " & TextLine
87                If TextLine <> "" Then
88                   If Split(Trim(TextLine), " ")(0) = "Private" Or
                        Split(Trim(TextLine), " ")(0) = "Function" Or
                        Split(Trim(TextLine), " ")(0) = "Sub" Then
89                      ReDim Preserve ArrDataModulB(vNo)
90                      ArrDataModulB(vNo).LineCode = i
91                      ArrDataModulB(vNo).ProcedureName = Trim(TextLine)
92                      vNo = vNo + 1
93                      SumProcedureT2.Text = UBound(ArrDataModulB) + 1
94                   End If
95                End If
96             End If
97             i = i + 1
98          End If
99       Loop
..     ……
..     ……
219 End Function
```

Gambar 5.6 Code Snippets Scanning dan Parser Source Code VB.Net

5.3.5 *Scanning* dan *parser* dalam mengenali struktur procedure VB.Net

Implementasi *scanning* dan *parser* dalam mengenali struktur procedure pada bahasa pemrograman VB.Net digambarkan pada Gambar 5.7. Baris kode ke-14 merupakan array bertipe data string yang digunakan untuk menampung isi dari kode program yang terdapat pada textbox SourceCodeB.Text yang merupakan hasil dai proses *scanning* dan *parser* pada sub bab 5.3.4.

Variable *vStringLine* yang terdapat pada baris kode ke 16-17 merupakan variable yang menampung baris kode yang sedang di-*scan* dengan terlebih dahulu menghilangkan angka yang terdapat diawal baris kode tersebut dengan bantuan fungsi *split* dengan karakter pemisah "||", variable *vStringLine* inilah yang kemudian akan dicek apakah syntax awal yang terdapat didalam isi dari variable *vStringLine* tersebut mengandung *string delimeter procedure.*

Jika string *delimeter procedure* untuk pendefinisian procedure terdapat diawal syntax *vStringLine*, maka isi dari variable *vStringLine* tersebut akan disimpan kedalam variable *sString* seperti pada baris kode ke-52, 56 dan baris kode lainnya yang terdapat variable *sString*.

Pengecekan yang dilakukan pada baris kode ke-36, 37 serta baris kode 64 dan 65 adalah untuk memeriksa apakah syntax awal dari isi variable *vStringLine* mengandung string *delimeter procedure* yang menyatakan akhir dari suatu procedure pada bahasa pemrograman VB.Net. Jika kondisi ini terpenuhi maka akan dilakukan proses filtering variable, filtering *stopword*, filtering tanda baca serta *casefolding* pada isi variable *sString*, selanjutnya isi variable *sString* disimpan kedalam sebuah array structure yang nantinya akan digunakan pada tahap pembandingan.

```
1   Private Sub CekModul_VBNET_Baseline()
2   Dim vStringLine As String
3   Dim vIndexProcedure As Byte
4   Dim vCekVariable As String
5   Dim sString As String
6
7   Dim lines() As String
8   Dim i As Integer
9   Dim j As Integer
10  Dim k As Integer
11  Dim l As Integer
12  Dim n As Integer
13
14  lines = Split(SourceCodeB.Text, vbCrLf)
15  For i = 0 To (UBound(lines) - 1)
16      vStringLine = Split(lines(i), "| ")(1)
17      vStringLine = Trim(vStringLine)
18      'jika membaca sebuah procedure maka abaikan
19      If Split(vStringLine, " ")(0) = "Public" Or
           Split(vStringLine, " ")(0) = "Private" Or
           Split(vStringLine, " ")(0) = "Function" Or
           Split(vStringLine, " ")(0) = "Sub" Or
           Split(vStringLine, " ")(0) = "Inherits" Then
20         ElseIf Mid(Split(vStringLine, " ")(0), 1, 1) = "'" Then
21                'Abaikan Comment
22         Else
23             If vSetFilterVariable = 1 Then
24                'Request proses menggunakan filtering variable
25                'Jika mengandung variable maka Simpan variable tersebut
26                'untuk kemudian digunakan sebagai filter
27                '===Cek Keberadaan variable
28                (... ... ... ... ...)
29                (... ... ... ... ...)
30                'Jika bukan Inisialisasi variable dan
31                'bukan pemanggilan sebuah procedure
32                'maka add vStringLine kedalam sString
33                If Split(vStringLine, " ")(0) <> "Dim" And
                     Split(vStringLine, " ")(0) <> "Call" Then
34                   'cek apakah akhir sebuah procedure jika "ya" maka
35                   'keluar jika "tidak" maka catat di sString
36                   If Split(Trim(vStringLine), " ")(0) = "End" Then
37                      If Split(Trim(vStringLine), " ")(1) = "Sub" Then
38                         '===Filtering variable
39                          (... ... ... ... ...)
40                         '===Case folding
41                          (... ... ... ... ...)
42                         '===Filtering StopWord
```

```
43                          (... ... ... ... ...)
44                        'Replace tanda baca
45                          (... ... ... ... ...)
46                        ArrDataModulB(vIndexProcedure).TextProcedure = sString
47                        vIndexProcedure = vIndexProcedure + 1
48                        'clear
49                        sString = ""
50                     Else
51                        vStringLine = Split(lines(i), "| ")(1)
52                        sString = Trim(sString) & " " & Trim(vStringLine)
53                     End If
54                  Else
55                     vStringLine = Split(lines(i), "| ")(1)
56                     sString = Trim(sString) & " " & Trim(vStringLine)
57                  End If
58               End If
59            Else
60               'Request proses tidak menggunakan filtering variable
61               If Split(vStringLine, " ")(0) <> "Dim" And
                    Split(vStringLine, " ")(0) <> "Call" Then
62                  'cek apakah akhir sebuah procedure jika "ya"
63                  'maka keluar jika "tidak" maka catat di textbox
64                  If Split(Trim(vStringLine), " ")(0) = "End" Then
65                     If Split(Trim(vStringLine), " ")(1) = "Sub" Then
66                        '===Case folding
67                          (... ... ... ... ...)
68                        '===Filtering StopWord
69                          (... ... ... ... ...)
70                        'Replace tanda baca
71                          (... ... ... ... ...)
72                        ArrDataModulB(vIndexProcedure).TextProcedure = sString
73                        vIndexProcedure = vIndexProcedure + 1
74                        'clear
75                        sString = ""
76                     Else
77                        vStringLine = Split(lines(i), "| ")(1)
78                        sString = Trim(sString) & " " & Trim(vStringLine)
79                     End If
80                  Else
81                     vStringLine = Split(lines(i), "| ")(1)
82                     sString = Trim(sString) & " " & Trim(vStringLine)
83                  End If
84               End If
85            End If
86      End If
87   Next
88   End Sub
```

Gambar 5.7 CodeSnippets Scanning dan Parser Struktur Procedure VB.Net

5.3.6 *Scanning* dan *parser variable* pada *source code* VB.Net

Proses *scanning* dan *parser* variable pada *source code* VB.Net merupakan bagian dari *code snippets* pada Gambar 5.7 diatas, yang terletak pada baris kode ke 28 dan 29, proses ini merupakan proses menganalisa keberadaan variable yang terdapat pada baris kode yang saat itu sedang di *scan.*

Dengan cara memisahkan kode program berdasarkan karakter *whitespace* (“ “), dan melakukan perulangan menggunakan “*for*”, kemudian melakukan

pengecekan satu persatu apakah variable vCekVariable = ”Dim” atau vCekVariable = ”Public”, dimana Dim dan Public adalah string delimeter untuk pendifinisian variable pada bahasa pemrograman VB.Net.

Jika kondisi diatas terpenuhi, maka nama variable akan ditemukan dengan bantuan kode `Split(vStringLine, " ")(j + 1)` seperti yang dijelaskan pada baris kode ke-5 pada Gambar 5.8, kemudian nama variable akan disimpan kedalam suatu array untuk keperluan filtering keberadaan variable pada text procedure. Proses *scanning* dan *parser* variable selengkapnya digambarkan pada Gambar 5.8.

```
1  For j = 0 To UBound(Split(vStringLine, " "))
2      vCekVariable = Split(vStringLine, " ")(j)
3      If Trim(vCekVariable) = "Dim" Or Trim(vCekVariable) = "Public" Then
4         ReDim Preserve ArrVariableModul1(n)
5         ArrVariableModul1(n) = Split(vStringLine, " ")(j + 1)
6         n = n + 1
7      End If
8  Next
```

Gambar 5.8 Code Snippets Scanning dan Parser Variable VB.Net

5.3.7 *Scanning* dan *parser source code* Delphi

Implementasi scanning dan parser pada *source code* Delphi digambarkan pada Gambar 5.9. Baris kode ke-102 merupakan kondisi pengecekan yang menyatakan apakah variable xExtention berisi “pas”, dimana “pas” merupakan ekstensi dari bahasa pemrograman Delphi, baris kode ke-110 menjadi parameter sistem untuk merekam data, ditandai dengan *keyword* “{$R *.dfm}”.

Pada baris kode ke-119 dan 138, saat proses *scanning* menemukan sebuah *procedure* atau *function* maka nama dari *procedure* atau *function* tersebut beserta no.baris akan ditampung kedalam sebuah *array structure*, dimana kumpulan *array* yang berisi nama-nama *procedure* atau *function* tadi akan dijadikan parameter untuk menyaring proses pemanggilan *procedure* yang mungkin saja terdapat pada *procedure* lainnya.

```
1   Function ReadSourceCode(ByVal xFilePath As String, ByVal xCode As String,
2   ByVal xExtention As String)
3   Dim TextLine As String
4   Dim i As Short = 1
5   Dim vNo As Short
6   Dim xCatat As Boolean
7
8   If Sistem.IO.File.Exists(xFilePath) = True Then
9         Dim objReader As New Sistem.IO.StreamReader(xFilePath)
10        Dim LineNumber As String
11        Cursor = Cursors.WaitCursor
..      ..............
```

```
     '+++++++++++++++++ Read Source Code Delphi ++++++++++++++++++++++++++
102    ElseIf xExtention = "pas" Then
103      Dim vTandaMulai As Boolean
104      Dim xAmbilString1 As String
105      Dim xAmbilString2 As String
106
107      Do While objReader.Peek() <> -1
108         TextLine = objReader.ReadLine()
109         TextLine = Trim(TextLine)
110         If TextLine = "{$R *.dfm}" Then
111            vTandaMulai = True
112         ElseIf vTandaMulai = True Then
113            LineNumber = i
114            If xCode = 1 Then
115               If Split(Trim(TextLine), " ")(0) = "procedure" Then
116                  SourceCodeA.Text = SourceCodeA.Text &
117                  (LineNumber.PadRight(5, "*")) & "|| " & TextLine & vbNewLine
118                  If Trim(TextLine) = "" Then
119                  ElseIf Split(Trim(TextLine), " ")(0) = "procedure" Then
120                     ReDim Preserve ArrDataModulA(vNo)
121                     ArrDataModulA(vNo).LineCode = i
122                     xAmbilString1 = Split(Trim(TextLine), " ")(1)
123                     xAmbilString2 = Split(xAmbilString1, "(")(0)
124                     ArrDataModulA(vNo).ProcedureName = Split(Trim(TextLine),
                        " ")(0) & " " & xAmbilString2
125                     vNo = vNo + 1
126                     SumProcedureT1.Text = UBound(ArrDataModulA) + 1
127                  End If
128                  xCatat = True
129                  i = i + 1
130               ElseIf xCatat = True Then
131                  SourceCodeA.Text = SourceCodeA.Text &
                     (LineNumber.PadRight(5, & "|| " & TextLine & vbNewLine
132                  i = i + 1
133               End If
134            Else
135               If Split(Trim(TextLine), " ")(0) = "procedure" Then
136                  SourceCodeB.Text = SourceCodeB.Text &
                     (LineNumber.PadRight(5,"*")) & "|| " & TextLine & vbNewLine
137                  If Trim(TextLine) = "" Then
138                  ElseIf Split(Trim(TextLine), " ")(0) = "procedure" Then
139                     ReDim Preserve ArrDataModulB(vNo)
140                     ArrDataModulB(vNo).LineCode = i
141                     xAmbilString1 = Split(Trim(TextLine), " ")(1)
142                     xAmbilString2 = Split(xAmbilString1, "(")(0)
143                     ArrDataModulB(vNo).ProcedureName = Split(Trim(TextLine),
                        " ")(0) & " " & xAmbilString2
144                     vNo = vNo + 1
145                     SumProcedureT2.Text = UBound(ArrDataModulB) + 1
146                  End If
147                  xCatat = True
148                  i = i + 1
149               ElseIf xCatat = True Then
150                  SourceCodeB.Text = SourceCodeB.Text &
                     (LineNumber.PadRight(5, & "|| " & TextLine & vbNewLine
151                  i = i + 1
152               End If
153            End If
154         End If
155      Loop
..     ..............
..     ..............
219  End Function
```

Gambar 5.9 Code Snippets Proses Scanning dan Parser Source Code Delphi

5.3.8 *Scanning* dan *parser* dalam mengenali struktur procedure Delphi

Implementasi *scanning* dan *parser* dalam mengenali struktur procedure pada bahasa pemrograman Delphi digambarkan pada Gambar 5.10. Baris kode ke-20 merupakan array bertipe data string yang digunakan untuk menampung isi dari kode program yang terdapat pada textbox SourceCodeB.Text yang merupakan hasil dai proses *scanning* dan *parser* pada sub bab 5.3.7.

Variable *vStringLine* yang terdapat pada baris kode ke 22-23 merupakan variable yang menampung baris kode yang sedang di-*scan* dengan terlebih dahulu menghilangkan angka yang terdapat diawal baris kode tersebut dengan bantuan fungsi *split* dengan karakter pemisah "||", variable *vStringLine* inilah yang kemudian akan dicek apakah syntax awal yang terdapat didalam isi dari variable *vStringLine* tersebut mengandung *string delimeter procedure.*

Jika string *delimeter procedure* untuk pendefinisian procedure terdapat diawal syntax *vStringLine,* maka isi dari variable *vStringLine* akan disimpan kedalam variable *xString1* seperti pada baris kode ke-166, 173 dan baris kode lainnya yang terdapat syntax `xString1 = Trim(xString1) & " " & Trim(vStringLine).`

Variable *xString1* sebenarnya merupakan suatu tempat yang disediakan untuk menampung syntax yang terdapat pada suatu *procedure* atau *function.* Jika *syntax* pada procedure telah ditampung kedalam variable ini maka proses selanjutnya akan dilakukan filtering variable, filtering *stopword,* filtering tanda baca serta *casefolding* pada isi variable *xString1,* selanjutnya isi variable *xString1* disimpan kedalam sebuah array structure yang nantinya akan digunakan ditahap pembandingan. Selengkapnya *code snippets* proses *scanner* dan *parser* dalam mengenali stuktur procedure delphi digambarkan pada Gambar 5.10.

```
1   Private Sub CekModul_Delphi_Baseline()
2   Dim vStringLine As String
3   Dim sString As String
4   Dim xString As String
5   Dim vTandaMasuk As Boolean
6   Dim vTandaKetemu As Boolean
7   Dim xKiriKotor As String
8   Dim xKiriBersih
9   Dim xString1 As String
10
11  Dim lines() As String
12  Dim i As Integer
13  Dim j As Integer
14  Dim k As Integer
```

```
15  Dim l As Integer
16  Dim m As Integer
17  Dim n As Integer
18  Dim x As Integer

19  Dim oregex As New RegularExpressions.Regex("[(\\\/.,:;+\*\?!""'<>|)]")
20  lines = Split(SourceCodeB.Text, vbCrLf)
21  For i = 0 To (UBound(lines) - 1)
22      vStringLine = Split(lines(i), "| ")(1)
23      vStringLine = Trim(vStringLine)
24      If vStringLine <> "" Then
25         'jika membaca sebuah procedure maka abaikan
26         If Split(vStringLine, " ")(0) = "procedure" Or
              Split(vStringLine, " ")(0) = "function" Then
27            If xString1 = "" Then
28            Else
29               sString = Replace(xString1, ";", "")
30               xString1 = ""
31               If vSetFilterVariable = 1 Then
32                  sString = oregex.Replace(sString, " ")
33                  sString = Trim(sString.Replace("  ", " "))
34                  sString = Trim(sString.Replace("   ", " "))
35                  sString = Trim(sString.Replace("    ", " "))
36                  sString = Trim(sString.Replace("     ", " "))
37                  'Cek tanda masuk
38                  For m = 0 To UBound(Split(sString, " "))
39                      If ArrVariableModul2 IsNot Nothing Then
40                        For k = 0 To UBound(ArrVariableModul2)
41                          If Split(sString," ")(m) = ArrVariableModul2(k) Then
42                             vTandaMasuk = True
43                          Exit For
44                          End If
45                        Next
46                      End If
47                      If vTandaMasuk = True And vTandaKetemu = False Then
48                      ElseIf vTandaKetemu = False And vTandaMasuk = False Then
49                            xString1 = xString1 & " " & Split(sString, " ")(m)
50                      End If
51                      vTandaMasuk = False
52                  Next
53               ElseIf vSetFilterVariable = 0 Then
54                  Dim vNmProce As String
55                  Dim vSubstring As String
56                  'cek pemanggilan sebuah procedure/function
57                  For m = 0 To UBound(Split(sString, " "))
58                      vSubstring = Split(sString, " ")(m)
59                      For k = 0 To UBound(ArrDataModulB)
60                      vNmProce = Split(ArrDataModulB(k).ProcedureName, " ")(1)
61                      If vSubstring = vNmProce Then
62                         Call ReplaceAll(sString, vNmProce, "")
63                      End If
64                      Next
65                  Next
66                  xString = LCase(Trim(sString))
67               End If
68                    '===Case folding
69                     (... ... ... ... ...)
70                    '===Filtering StopWord
71                     (... ... ... ... ...)
72                    'Replace tanda baca
73                     (... ... ... ... ...)
74                    ArrDataModulB(x).TextProcedure = xString1
75                    x = x + 1
76                    xString1 = ""
```

```
77              End If
78          ElseIf Mid(Split(vStringLine, " ")(0), 1, 2) = "//" Then
79                'Abaikan Comment
80          Else
81              If vSetFilterVariable = 1 Then
82                 'Request proses menggunakan filtering variable
83                 'Jika mengandung variable maka Simpan variable tersebut
84                 'untuk kemudian digunakan sebagai filter
85                 '===Cek keberadaan variable
86                  (... ... ... ... ...)
87                  (... ... ... ... ...)
88                  '====Jika bukan Inisialisasi variable dan
89                  'bukan pemanggilan sebuah procedure
90                  'maka add vStringLine kedalam xString1
91                  If Split(vStringLine, " ")(0) <> "var" And
                       Split(vStringLine, " ")(0) <> "Call" Then
92                     'cek apakah akhir sebuah procedure jika "ya" maka
93                     'keluar jika "tidak" maka catat di textbox
94                     If Split(Trim(vStringLine), " ")(0) = "procedure" Or
                          Split(Trim(vStringLine), " ")(0) = "function" And
                          xString1 <> "" Then
95                        sString = oregex.Replace(xString1, " ")
96                        sString = Trim(sString.Replace("  ", " "))
97                        sString = Trim(sString.Replace("   ", " "))
98                        sString = Trim(sString.Replace("    ", " "))
99                        sString = Trim(sString.Replace("     ", " "))
100                       xString1 = ""
101                       'cek tanda masuk
102                       For m = 0 To UBound(Split(sString, " "))
103                       If ArrVariableModul2 IsNot Nothing Then
104                          For k = 0 To UBound(ArrVariableModul2)
105                          If Split(sString," ")(m) = ArrVariableModul2(k) Then
106                             vTandaMasuk = True
107                          Exit For
108                          End If
109                          Next
110                       End If
111                       If vTandaMasuk = True And vTandaKetemu = False Then
112                       ElseIf vTandaKetemu = False And vTandaMasuk=False Then
113                       xString1 = xString1 & " " & Split(sString, " ")(m)
114                       End If
115                       vTandaMasuk = False
116                       Next
117                    '===Case folding
118                       (... ... ... ... ...)
119                    '===Filtering StopWord
120                       (... ... ... ... ...)
121                    'Replace tanda baca
122                       (... ... ... ... ...)
123                       ArrDataModulB(x).TextProcedure = xString1
124                       x = x + 1
125                       xString1 = ""
126                 '==== Jika akhir dari modul =====
127                 ElseIf Split(Trim(vStringLine), " ")(0) = "end." And
128                        xString1 <> "" Then
129                       sString = oregex.Replace(xString1, " ")
130                       sString = Replace(sString, "(", " ")
131                       sString = Replace(sString, ")", " ")
132                       sString = Trim(sString.Replace("  ", " "))
133                       sString = Trim(sString.Replace("   ", " "))
134                       sString = Trim(sString.Replace("    ", " "))
135                       sString = Trim(sString.Replace("     ", " "))
136                       xString1 = ""
137                       'Cek tanda masuk
```

```
138                     For m = 0 To UBound(Split(sString, " "))
139                        If ArrVariableModul2 IsNot Nothing Then
140                          For k = 0 To UBound(ArrVariableModul2)
141                          If Split(sString," ")(m) = ArrVariableModul2(k) Then
142                             vTandaMasuk = True
143                             Exit For
144                          End If
145                          Next
146                        End If
147                        If vTandaMasuk = True And vTandaKetemu = False Then
148                        ElseIf vTandaKetemu = False And vTandaMasuk=False Then
149                           xString1 = xString1 & " " & Split(sString, " ")(m)
150                        End If
151                        vTandaMasuk = False
152                     Next
153                       '===Case folding
154                        (... ... ... ... ...)
155                       '===Filtering StopWord
156                        (... ... ... ... ...)
157                       'Replace tanda baca
158                        (... ... ... ... ...)
159                        ArrDataModulB(x).TextProcedure = xString1
160                        x = x + 1
161                        xString1 = ""
162                   Else
163                       vStringLine = Replace(Trim(vStringLine), ":=", " ")
164                       vStringLine = Replace(Trim(vStringLine), "=", " ")
165                       vStringLine = Replace(Trim(vStringLine), "<>", " ")
166                       xString1 = Trim(xString1) & " " & Trim(vStringLine)
167                   End If
168               ElseIf Split(Trim(vStringLine), " ")(0) = "var" Then
169               Else
170                   vStringLine = Replace(Trim(vStringLine), ":=", " ")
171                   vStringLine = Replace(Trim(vStringLine), "=", " ")
172                   vStringLine = Replace(Trim(vStringLine), "<>", " ")
173                   xString1 = Trim(xString1) & " " & Trim(vStringLine)
174               End If
175           ElseIf vSetFilterVariable = 0 Then
176               If Split(vStringLine, " ")(0) <> "var" And
                     Split(vStringLine, " ")(0) <> "Call" Then
177                  'cek apakah akhir sebuah procedure jika "ya" maka keluar
178                  'jika "tidak" maka catat di sString1
179                   If Split(Trim(vStringLine), " ")(0) = "procedure" Or
                        Split(Trim(vStringLine), " ")(0) = "function" And
                        xString1 <> "" Then
180                    '===Case folding
181                     (... ... ... ... ...)
182                    '===Filtering StopWord
183                     (... ... ... ... ...)
184                    'Replace tanda baca
185                     (... ... ... ... ...)
186                     ArrDataModulB(x).TextProcedure = xString1
187                     x = x + 1
188                     xString1 = ""
189                  '=====Jika akhir dari modul=======
190                  ElseIf Split(Trim(vStringLine), " ")(0) = "end." And
                           xString1 <> "" Then
191                        '===Case folding
192                         (... ... ... ... ...)
193                        '===Filtering StopWord
194                         (... ... ... ... ...)
195                        'Replace tanda baca
196                         (... ... ... ... ...)
197                         ArrDataModulB(x).TextProcedure = xString1
```

```
198                             x = x + 1
199                             xString1 = ""
200                     Else
201                         vStringLine = Replace(Trim(vStringLine), ":=", " ")
202                         vStringLine = Replace(Trim(vStringLine), "=", " ")
203                         vStringLine = Replace(Trim(vStringLine), "<>", " ")
204                         xString1 = Trim(xString1) & " " & Trim(vStringLine)
205                     End If
206                 ElseIf Split(Trim(vStringLine), " ")(0) = "var" Then
207                 Else
208                     vStringLine = Replace(Trim(vStringLine), ":=", " ")
209                     vStringLine = Replace(Trim(vStringLine), "=", " ")
210                     vStringLine = Replace(Trim(vStringLine), "<>", " ")
211                     xString1 = Trim(xString1) & " " & Trim(vStringLine)
212                 End If
213             End If
214         End If
215     End If
216 Next
217 End Sub
```

Gambar 5.10 CodeSnippets Scanning dan Parser Struktur Procedure Delphi

5.3.9 *Scanning* dan *parser* variable pada *source code* Delphi

Proses *scanning* dan *parser* variable pada *source code* Delphi merupakan bagian dari *code snippets* pada Gambar 5.10 diatas, yang terletak pada baris kode ke 86 dan 87, proses ini merupakan proses menganalisa keberadaan variable yang terdapat pada baris kode yang saat itu sedang di *scan.* dengan melakukan pengecekan terhadap isi dari variable *vStringLine.*

Jika diawal syntax variable *vStringLine* terdapat string *delimeter variable*, maka nama variable akan didapatkan dengan bantuan fungsi *split* dan *replace* seperti pada *code snippets* proses *scanning* dan *parser* variable yang selengkapnya digambarkan pada Gambar 5.11.

```
1   If Split(vStringLine, " ")(0) = "var" Then
2      xKiriKotor = Split(vStringLine, ":")(0)
3      xKiriBersih = Replace(xKiriKotor, "var", "")
4      Call Replace(Trim(xKiriBersih), ",", " ")
5      ReDim Preserve ArrVariableModul1(n)
6      ArrVariableModul1(n) = Trim(xKiriBersih)
7      n = n + 1
8   End If
```

Gambar 5.11 Code Snippets Scanning dan Parser Variable Delphi

5.3.10 *Scanning* dan *parser source code* C#

Implementasi *scanning* modul *source code* C# digambarkan pada Gambar 5.12. Baris kode ke-158 merupakan kondisi pengecekan yang menyatakan apakah variable xExtention berisi “cs” dimana “cs” merupakan ekstensi dari bahasa pemrograman C#, proses pembacaan *source code* dilakukan secara *line by line*

yang ditunjukkan pada baris kode ke-161, baris kode ke-164 dan 189 menjadi parameter sistem untuk menulis data, diawali dengan *keyword* "*namespace*" yang menyatakan suatu *Class* pada modul, atau diawali dengan *keyword "Using"* yang biasa digunakan untuk menambahkan *library* yang akan dipakai.

Pada baris kode ke-173 dan 198, dimana saat proses *scanning* menemukan sebuah *procedure* atau *method* maka nama dari *procedure* atau *method* tersebut beserta no.baris kode-nya akan ditampung kedalam sebuah struktur data yang berupa *array structure*, dimana kumpulan array yang berisi nama-nama *procedure* atau *method* tadi akan dijadikan parameter untuk menyaring proses pemanggilan procedure yang mungkin saja akan ada pada procedure lainnya.

```
1    Function ReadSourceCode(ByVal xFilePath As String, ByVal xCode As String,
2    ByVal xExtention As String)
3    Dim TextLine As String
4    Dim i As Short = 1
5    Dim vNo As Short
6    Dim xCatat As Boolean
7
8    If Sistem.IO.File.Exists(xFilePath) = True Then
9          Dim objReader As New Sistem.IO.StreamReader(xFilePath)
10         Dim LineNumber As String
11         Cursor = Cursors.WaitCursor
..       ..............
..       ..............
       '++++++++++++++++++++ Read Source Code C# ++++++++++++++++++++
158    ElseIf xExtention = "cs" Then
159      Dim nString As String
160      Do While objReader.Peek() <> -1
161         nString = objReader.ReadLine() & vbNewLine
162         TextLine = Trim(nString)
163         If xCode = 1 Then
164            If Split(Trim(TextLine), " ")(0) = "using" Or
                  Split(Trim(TextLine), " ")(0) = "namespace" Then
165               xCatat = True
166               LineNumber = i
167               SourceCodeA.Text = SourceCodeA.Text & (LineNumber.PadRight(5,
                  "*")) & "|| " & TextLine
168               i = i + 1
169            ElseIf xCatat = True Then
170            TextLine = Trim(nString)
171            LineNumber = i
172               If Trim(TextLine) = "" Then
173               ElseIf Split(Trim(TextLine), " ")(0) = "private" Or
                     Split(Trim(TextLine), " ")(0) = "public" Then
174                  If Split(Trim(TextLine), " ")(1) = "void" Then
175                     ReDim Preserve ArrDataModulA(vNo)
176                     ArrDataModulA(vNo).LineCode = i
177                     ArrDataModulA(vNo).ProcedureName = Trim(TextLine)
178                     vNo = vNo + 1
179                     SumProcedureT1.Text = UBound(ArrDataModulA) + 1
180                     SourceCodeA.Text = SourceCodeA.Text &
181                     (LineNumber.PadRight(5,"*")) & "|| " & TextLine
182                  End If
183               Else
```

```
184                     SourceCodeA.Text = SourceCodeA.Text &
                                           (LineNumber.PadRight(5, "*"))
                                            & "|| " & TextLine
185                 End If
186                 i = i + 1
187              End If
188           Else
189              If Split(Trim(TextLine), " ")(0) = "using" Or
                    Split(Trim(TextLine), " ")(0) = "namespace" Then
190                 xCatat = True
191                 LineNumber = i
192                 SourceCodeB.Text = SourceCodeB.Text & (LineNumber.PadRight(5,
                                       "*")) & "|| " & TextLine
193                 i = i + 1
194              ElseIf xCatat = True Then
195                 TextLine = Trim(nString)
196                 LineNumber = i
197                 If Trim(TextLine) = "" Then
198                 ElseIf Split(Trim(TextLine), " ")(0) = "private" Or
                       Split(Trim(TextLine), " ")(0) = "public" Then
199                    If Split(Trim(TextLine), " ")(1) = "void" Then
200                       ReDim Preserve ArrDataModulB(vNo)
201                       ArrDataModulB(vNo).LineCode = i
202                       ArrDataModulB(vNo).ProcedureName = Trim(TextLine)
203                       vNo = vNo + 1
204                       SumProcedureT2.Text = UBound(ArrDataModulA) + 1
205                       SourceCodeB.Text = SourceCodeB.Text &
                          (LineNumber.PadRight(5, "*")) & "|| " & TextLine
206                    End If
207                 Else
208                    SourceCodeB.Text = SourceCodeB.Text &
                       (LineNumber.PadRight(5,"*")) & "|| " & TextLine
209                 End If
210                 i = i + 1
211              End If
212           End If
213        Loop
214     End If
215     Cursor = Cursors.Default
216  Else
217        MsgBox("File Does Not Exist")
218  End If
219  End Function
```

Gambar 5.12 Code Snippets Proses Scanning dan Parser Source Code C#

5.3.11 *Scanning* dan *parser* dalam mengenali struktur procedure C#

Implementasi *scanning* dan *parser* dalam mengenali struktur procedure pada bahasa pemrograman C# digambarkan pada Gambar 5.13. Baris code ke-20 merupakan array bertipe data string yang digunakan untuk menampung isi dari kode program yang terdapat pada textbox SourceCodeB.Text yang merupakan hasil dai proses *scanning* dan *parser* pada sub bab 5.3.10.

Variable *vStringLine* yang terdapat pada baris kode ke 26 merupakan variable yang menampung baris kode yang sedang di-*scan* dengan terlebih dahulu menghilangkan angka yang terdapat diawal baris kode tersebut dengan bantuan

fungsi *split* dengan karakter pemisah "||", isi variable *vStringLine* inilah yang kemudian akan dicek apakah terdapat *string delimeter procedure* pada awal *syntax* nya.

Beberapa string *delimeter procedure* seperti "*private*", "*public*" digunakan untuk mendeteksi keberadaan suatu procedure pada baris kode yang sedang di *scan* seperti yang terlihat pada baris kode ke-32, 33 dan baris kode lainnya.

Pada baris kode ke-143, 168 dan baris kode lainnya yang terdapat syntax `xString = Trim(xString) & " " & Trim(vStringLine)` dimana variable *xString* sebenarnya merupakan suatu tempat yang disediakan untuk menampung syntax yang terdapat pada suatu *procedure* atau *function* dan jika suatu syntax pada procedure telah ditampung kedalam variable ini maka proses selanjutnya akan dilakukan filtering variable, filtering *stopword*, filtering tanda baca serta *casefolding* pada isi variable *xString*, selanjutnya isi variable *xString* disimpan kedalam sebuah array structure yang nantinya akan digunakan ditahap pembandingan. Selengkapnya *code snippets* proses scanner dan parser dalam mengenali stuktur procedure C# digambarkan pada Gambar 5.13.

```
1   Private Sub CekModul_CSharp_Baseline()
2   Dim vStringLine As String
3   Dim sString As String
4   Dim xString As String
5   Dim vTandaMasuk As Boolean
6   Dim vTandaKetemu As Boolean
7   Dim vBolehDiCatat As Boolean
8
9   Dim lines() As String
10  Dim i As Integer
11  Dim j As Integer
12  Dim k As Integer
13  Dim l As Integer
14  Dim m As Integer
15  Dim n As Integer
16  Dim x As Integer
17  Dim nString As String
18
19  Dim oregex As New RegularExpressions.Regex("[(\\\/.,:;+\*\?!""'<>|)]")
20  lines = Split(SourceCodeB.Text, vbCrLf)
21  For i = 0 To (UBound(lines) - 1)
22      nString = Split(lines(i), "| ")(1)
23      nString = Replace(nString, "       ", "")
24      nString = Replace(nString, "               ", "")
25      nString = Replace(nString, "                       ", "")
26      vStringLine = Trim(nString)
27      If Trim(vStringLine) = "{" Or Trim(vStringLine) = "}" Then
28         vStringLine = ""
29      End If
30      If vStringLine <> "" Then
31         'jika membaca sebuah procedure maka catata kedalam text box
```

```
32          If Split(Trim(vStringLine), " ")(0) = "private" Or
33             Split(vStringLine, " ")(0) = "public" Then
34             If Split(vStringLine, " ")(1) = "void" And
35                vBolehDiCatat = False Then
36                vBolehDiCatat = True
37             ElseIf Split(vStringLine, " ")(1) = "void" And
38                    vBolehDiCatat = True Then
39                    vBolehDiCatat = False
40             End If
41             If Split(Trim(vStringLine), " ")(0) = "private" Or
42                Split(Trim(vStringLine), " ")(0) = "public" Then
43                If Split(Trim(vStringLine), " ")(1) = "void" And
44                   xString <> "" And vBolehDiCatat = False Then
45                   vBolehDiCatat = True
46                   If vSetFilterVariable = 0 Then
47                      '===Case folding
48                       (... ... ... ... ...)
49                      '===Filtering StopWord
50                       (... ... ... ... ...)
51                      'Replace tanda baca
52                       (... ... ... ... ...)
53                      ArrDataModulB(x).TextProcedure = xString
54                      x = x + 1
55                      xString = ""
56                      ElseIf vSetFilterVariable = 1 Then
57                             sString = xString
58                             sString = oregex.Replace(sString, " ")
59                             xString = ""
60                             For m = 0 To UBound(Split(sString, " "))
61                                 If ArrVariableModul2 IsNot Nothing Then
62                                    For k = 0 To UBound(ArrVariableModul2)
63                                       If Split(sString, " ")(m) =
64                                          ArrVariableModul2(k) Then
65                                          vTandaMasuk = True
66                                       Exit For
67                                       End If
68                                    Next
69                                 End If
70                                 If vTandaMasuk = True And
71                                    vTandaKetemu = False Then
72                                 ElseIf vTandaKetemu = False And
73                                        vTandaMasuk = False Then
74                                        xString = Trim(xString) & " " &
75                                        Trim(Split(sString, " ")(m))
76                                 End If
77                                 vTandaMasuk = False
78                             Next
79                             '===Case folding
80                              (... ... ... ... ...)
81                             '===Filtering StopWord
82                              (... ... ... ... ...)
83                             'Replace tanda baca
84                              (... ... ... ... ...)
85                              xString = Replace(xString, " ", "")
86                              ArrDataModulB(x).TextProcedure = xString
87                              x = x + 1
88                              xString = ""
89                      End If
90                   End If
91                End If
92             ElseIf Mid(Split(vStringLine, " ")(0), 1, 2) = "//" Then
93          ElseIf vBolehDiCatat = True Then
94             If vSetFilterVariable = 1 Then
95                'Request proses menggunakan filtering variable
```

```
96                'Jika mengandung variable maka Simpan variable tersebut
97                'untuk kemudian digunakan sebagai filter
98                '===Cek keberadaan variable
99                 (... ... ... ... ...)
100                (... ... ... ... ...)
101               If Split(vStringLine, " ")(0) <> "string" And
102                  Split(vStringLine, " ")(0) <> "int" Then
103                  'cek apakah akhir sebuah procedure jika "ya"
104                  'maka keluar jika "tidak" maka catat di textbox
105                  If Split(Trim(vStringLine), " ")(0) = "private" Or
106                     Split(Trim(vStringLine), " ")(0) = "public" And
107                     xString <> "" Then
108                     sString = xString
109                     sString = oregex.Replace(sString, " ")
110                     xString = ""
111                     For m = 0 To UBound(Split(sString, " "))
112                        For k = 0 To UBound(ArrVariableModul2)
113                            If Split(sString, " ")(m) =
114                               ArrVariableModul2(k) Then
115                               vTandaMasuk = True
116                            Exit For
117                            End If
118                        Next
119                        If vTandaMasuk = True And vTandaKetemu = False Then
120                        ElseIf vTandaKetemu = False And
121                               vTandaMasuk = False Then
122                               xString = Trim(xString) & " " &
123                               Trim(Split(sString, " ")(m))
124                        End If
125                        vTandaMasuk = False
126                     Next
127                     '===Case folding
128                      (... ... ... ... ...)
129                     '===Filtering StopWord
130                      (... ... ... ... ...)
131                     'Replace tanda baca
132                      (... ... ... ... ...)
133                      xString = Replace(xString, " ", "")
134                      ArrDataModulB(x).TextProcedure = xString
135                      x = x + 1
136                      xString = ""
137                   ElseIf Split(Trim(vStringLine), " ")(0) = "string" Or
138                          Split(Trim(vStringLine), " ")(0) = "int" Then
139                      Else
140                          vStringLine = Replace(Trim(vStringLine), ":=", " ")
141                          vStringLine = Replace(Trim(vStringLine), "=", " ")
142                          vStringLine = Replace(Trim(vStringLine), "<>", " ")
143                          xString = Trim(xString) & " " & Trim(vStringLine)
144                      End If
145                End If
146             ElseIf vSetFilterVariable = 0 Then
147                If Split(vStringLine, " ")(0) <> "var" And
148                   Split(vStringLine, " ")(0) <> "Call" Then
149                   'cek apakah akhir sebuah procedure jika "ya"
150                   'maka keluar jika "tidak" maka catat di textbox
151                   If Split(Trim(vStringLine), " ")(0) = "private" Or
152                      Split(Trim(vStringLine), " ")(0) = "public" And
153                      xString <> "" Then
154                     '===Case folding
155                      (... ... ... ... ...)
156                     '===Filtering StopWord
157                      (... ... ... ... ...)
158                     'Replace tanda baca
159                      (... ... ... ... ...)
```

```
160                     xString = Replace(xString, " ", "")
161                     ArrDataModulB(x).TextProcedure = xString
162                     x = x + 1
163                     xString = ""
164                 ElseIf vBolehDiCatat = True Then
165                     vStringLine = Replace(Trim(vStringLine), ":=", " ")
166                     vStringLine = Replace(Trim(vStringLine), "=", " ")
167                     vStringLine = Replace(Trim(vStringLine), "<>", " ")
168                     xString = Trim(xString) & " " & Trim(vStringLine)
169                 End If
170             End If
171         End If
172     End If
173 End If
174     'cek apakah akhir dari procedure
175     If i = (UBound(lines) - 1) And xString <> "" Then
176         If vSetFilterVariable = 1 Then
177             sString = xString
178             sString = oregex.Replace(sString, " ")
179             xString = ""
180             For m = 0 To UBound(Split(sString, " "))
181                 For k = 0 To UBound(ArrVariableModul2)
182                     If Split(sString, " ")(m) = ArrVariableModul2(k) Then
183                         vTandaMasuk = True
184                     Exit For
185                     End If
186                 Next
187                 If vTandaMasuk = True And vTandaKetemu = False Then
188                 ElseIf vTandaKetemu = False And vTandaMasuk = False Then
189                 xString = Trim(xString) & " " & Trim(Split(sString," ")(m))
190                 End If
191                 vTandaMasuk = False
192             Next
193             '===Case folding
194              (... ... ... ... ...)
195             '===Filtering StopWord
196              (... ... ... ... ...)
197             'Replace tanda baca
198              (... ... ... ... ...)
199             xString = Replace(xString, " ", "")
200             ArrDataModulB(x).TextProcedure = xString
201             x = x + 1
202             xString = ""
203         ElseIf vSetFilterVariable = 0 Then
204             '===Case folding
205              (... ... ... ... ...)
206             '===Filtering StopWord
207              (... ... ... ... ...)
208             'Replace tanda baca
209              (... ... ... ... ...)
210             xString = Replace(xString, " ", "")
211             ArrDataModulB(x).TextProcedure = xString
212             x = x + 1
213             xString = ""
214         End If
215     End If
216 Next
217 End Sub
```

Gambar 5.13 Code Snippets Scanning dan Parser Struktur Procedure C#

5.3.12 *Scanning* dan *parser variable* pada *source code* C#

Proses *scanning* dan *parser* variable pada *source code* C# merupakan bagian dari *code snippets* pada Gambar 5.13 diatas, yang terletak pada baris kode ke 99 dan 100, proses ini merupakan proses menganalisa keberadaan variable yang terdapat pada baris kode yang saat itu sedang di *scan*, dengan cara melakukan pengecekan terhadap isi dari variable *vStringLine*.

Jika syntax awal variable *vStringLine* mengandung suatu delimeter variable maka nama variable akan mudah didapatkan dengan bantuan syntax `Split(vStringLine, " ")(1)`, kemudian nama variable tersebut disimpan kedalam suatu array untuk selanjutnya akan digunakan sebagai filtering keberadaan variable pada tahap selanjutnya. *Code snippets* scanning dan filtering variable digambarkan pada Gambar 5.14.

```
1  If Split(Trim(vStringLine), " ")(0) = "string" Or
2     Split(Trim(vStringLine), " ")(0) = "int" Or
3     Split(Trim(vStringLine), " ")(0) = "double" Then
4         ReDim Preserve ArrVariableModul1(n)
5         ArrVariableModul1(n) = Split(vStringLine, " ")(1)
6         n = n + 1
7  End If
```

Gambar 5.14 Code Snippets Scanning dan Parser Variable C#

5.4 Filtering Variable

Filtering variable adalah proses penghapusan variable-variable yang terdapat dalam sebuah procedure, filtering variable dilakukan ketika opsi pengecekan filter variable diaktifkan oleh pengguna, proses filtering variable dilakukan dengan melakukan iterasi terhadap sejumlah variable modul yang terdapat pada array "ArrVariableModul2" yang telah didapat dari hasil *scanning*, setiap proses iterasi yang dilakukan akan memanggil function "ReplaceAll".

Proses pemanggilan function "ReplaceAll" disertakan dengan pengiriman tiga variable yaitu variable "sString" yang berisi *syntax* yang sedang diproses, variable "ReplaceThis" yang berisi sekumpulan variable hasil *scanning source code*, serta variable "WithThis" yang tidak berisi karakter yang akan digunakan untuk melakukan *replace* terhadap *syntax* yang memiliki variable. Hasil replace akan dikembalikan ke variable "sString" untuk diproses ketahap *case-folding*, *Syntax* filtering variable dapat dilihat pada Gambar 5.15.

```
If Not ArrVariableModul2 Is Nothing Then
    For k = 0 To UBound(ArrVariableModul2)
       Call ReplaceAll(sString, ArrVariableModul2(k), "")
    Next
End If
```

```
Function ReplaceAll(ByRef sString As String, ByVal ReplaceThis As String, ByVal
WithThis As String)
    Dim Temp As Object
    Temp = Split(sString, ReplaceThis)
    sString = Join(Temp, WithThis)
End Function
```

Gambar 5.15 Code snippets proses filtering variable

5.5 Filtering *Stop-Word*

Proses filtering *stop-word* dilakukan untuk menghapus *keyword-keyword* tertentu yang tidak ingin digunakan pada proses pembentukan token procedure. Pada sistem yang dirancang ini *keyword-keyword* tersebut tersimpan didalam sebuah file yang berekstensi ".txt" yang nantinya isi dari file tersebut yang berupa *keyword stopword* akan dibaca oleh sistem sesuai dengan ekstensi dari bahasa pemrograman yang sedang di *scan*, dan *keyword* tersebut akan ditampung kedalam sebuah array untuk digunakan sebagai filtering terhadap *syntax* yang sedang di *scanning*.

Pada Gambar 5.16 proses *filtering stopword* diawali dengan membaca array "ArrFilter" yang berisi *keyword stopword*. Pada setiap iterasi yang dilakukan akan memanggil sebuah function yang bernama "FilterStopWord" yang diikuti dengan pengiriman dua parameter yang berupa "xString" yang berisi *syntax* yang sedang dibaca dan *keyword* yang terdapat pada proses iterasi array pertama.

```
For l = 0 To UBound(ArrFilter)
        Call FilterStopWord(sString, ArrFilter(l), "")
Next
```

```
Function FilterStopWord(ByRef xString As String, ByVal ReplaceThis As String,
ByVal WithThis As String)
        Dim Temp As Object
        Temp = Split(xString, ReplaceThis)
        xString = Join(Temp, WithThis)
End Function
```

Gambar 5.16 Code snippets proses filtering stop-word

5.6 Filtering Tanda-baca

Filtering tanda-baca adalah proses penghapusan tanda baca yang ada pada *syntax* yang terdapat pada variable *xText*, dimana *xText* berisi sebuah *syntax* dari hasil *scanning* dari suatu procedure atau function yang dilakukan oleh sistem. Proses filtering dilakukan dengan memanggil procedure "pReplace" dimana

procedure tersebut akan menghapus tanda-baca yang ada pada *xText* dengan menggunakan method *Replace* yang ada pada bahasa pemrograman Visual Basic.Net. *Function* filtering tanda baca dapat dilihat pada Gambar 5.17.

```
1   Function pReplace(ByRef xText As String)
2       xText = Replace(xText, " ", "")
3       xText = Replace(xText, ".", " ")
4       xText = Replace(xText, "'", " ")
5       xText = Replace(xText, ",", " ")
6       xText = Replace(xText, "&", " ")
7       xText = Replace(xText, "(", " ")
8       xText = Replace(xText, ")", " ")
9       xText = Replace(xText, "{", " ")
10      xText = Replace(xText, "}", " ")
11      xText = Replace(xText, "=", " ")
12      xText = Replace(xText, "_", " ")
13      xText = Replace(xText, ";", " ")
14      xText = Replace(xText, """", " ")
15  End Function
```

Gambar 5.17 Code snippets proses filtering tanda-baca

5.7 *Case Folding*

Proses *case folding* adalah proses merubah keseluruhan *syntax* menjadi huruf kecil. Proses ini dilakukan dengan memanfaatkan fungsi *ToLower* yang ada pada bahasa pemrograman Visual Basic. *syntax* untuk merubah keseluruhan karakter menjadi huruf kecil dapat dilihat pada Gambar 5.18.

```
1   sString = Trim(sString.ToLower)
```

Gambar 5.18 Code snippets proses *case folding*

5.8 Hashing

Proses *hashing* adalah proses merubah token string menjadi token yang berbentuk angka, setiap karakter yang ada pada token string akan diterjemahkan kode *ascii* nya untuk dilakukan perhitungan nilai hash pada token string tersebut, proses *hashing* digambarkan pada Gambar 5.19.

```
1   For x = 0 To Len(TextStringT) - S
2       ReDim Preserve ArrTokenString(x)
3       ArrTokenString(x).xTokenStringT = Mid(TextStringT.ToLower, increment, S)
4       ArrSubStringT = ArrTokenString(x).xTokenStringT.ToLower
5       n = Len(ArrTokenString(x).xTokenStringT)
6       A1 = Asc(Microsoft.VisualBasic.Left(ArrSubStringT, 1))
7       An = Asc(Microsoft.VisualBasic.Right(ArrSubStringT, 1))
8       xTokenAscii = A1 + An + (n * Zz)
9       ArrTokenString(x).xHashValue = xTokenAscii
10      increment = increment + 1
11  Next
```

Gambar 5.19 Code snippets proses hashing

5.9 Proses Pengecekan Kemiripan Suatu Procedure

Proses pengecekan kemiripan string antar kedua *procedure* yang dibandingkan dilakukan menggunakan algoritma Running Karp-Rabin Greedy String Tiling. Pada top level algoritma RKRGST terdapat fase *scanpattern* untuk mencari string yang sama antar kedua procedure menggunakan *hash value* dan fase markstring dilakukan untuk menandai string yang memiliki kesamaan dari hasil *scanpattern*.

5.9.1 Top level algoritma RKRGST

Top level algoritma RKRGST memiliki variable *search length (s)* dimana variable *search length* tersebut digunakan untuk membentuk pasangan substring P[p..p+s-1] dengan p antara 1 sampai |p|-s. Pada baris kode ke-5 nilai s di inisialisasikan = 4 dan Minimum Match Length (MML) = s.

Pada baris kode ke-9 terdapat pemanggilan procedure *scanpattern* dengan menyertakan tiga parameter yaitu S, TextstringP dan TextstringT, dimana *s* merupakan panjang substring yang akan dibentuk dari textstringP dan textstringT, sedangkan textstringP dan textstringT sendiri merupakan *syntax* dari 2 (dua) procedure yang akan dibandingkan. Top level algoritma RKRGST digambarkan pada Gambar 5.20.

```
1  Private Sub TopLevelRKRGST(ByVal TextStringP As String, ByVal TextStringT As
2  String)
3  Try
4     Dim S As Integer ' Search Length
5     S = 4
6     MML = S
7     Do
8        vExitLoop = False
9        Call ScanPattern(S, TextStringP, TextStringT)
10       If LMax > (2 * S) Then
11          S = LMax
12       Else
13           Call Markstring(S)
14           If S > (2 * MML) Then
15              S = S \ 2
16           ElseIf S > MML Then
17              S = MML
18           Else
19              If S = 0 And xKondisiMark = False Then
20                 Length_Of_Token_Tile = 0
21                 Call xReleaseArray()
22                 Erase xStatusMark_T
23                 LMax = 0
24                 Exit Sub
25              Else
26                 'next step baca procedure selanjutnya
27                 Length_Of_Token_Tile = 0
28                 xKondisiMark = False
29                 Call BebaskanMemory()
```

```
30                  vExitLoop = True
31                  LMax = 0
32                  Exit Sub
33              End If
34          End If
35      End If
36   Loop Until S < MML
37      Call BebaskanMemory()
38 Catch ex As Exception
39    MsgBox(ex.Message, MsgBoxStyle.OkOnly, "Error Top Level RKRGST")
40 End Try
41 End Sub
```

Gambar 5.20 Code snippets top level algoritma RKRGST

Setiap kali pemanggilan procedure *scanpattern* terdapat nilai kembalian terbesar dari string yang cocok dari hasil *scanning* yang dilakukan. Pada *source code* diatas nilai tersebut disimpan dalam variable LMax, kemudian pada baris kode ke-10 dilakukan pengecekan apakah nilai LMax > (2*s) jika ya, maka set s=LMax dan lakukan *scanpattern* dengan nilai *s* yang baru. Jika LMax < (2*s) maka panggil procedure markstring untuk melakukan penandaan pada *list-of-maximal-match* yang terbentuk dari hasil *scanpattern*.

5.9.2 Fase *scanpattern*

Fase *Scanpattern* di-implementasikan kedalam bentuk *function* dengan menerima 3 parameter yang dikirim dari Top level algoritma RKRGST. Parameter tersebut berupa *search length* yang dituliskan dalam variable (*s*) kemudian TextStringP dan TextStringT. Dimana TextStringP dan TextStringT merupakan *syntax* procedure yang pada saat itu sedang di *scan* dari masing-masing *source code* untuk dibandingkan kemiripannya.

Pada baris kode ke-19 dilakukan pengecekan apakah kondisi karakter pada TextStringT telah ada yang tertandai, jika ada satu saja karakter yang telah tertandai maka, cek jarak posisi indeks yang ada pada saat itu ke *tile* berikutnya, apabila lebih kecil dari *search-length (s)*, maka bagian yang belum tertandai pada TextStringT mulai dari posisi indeks tersebut akan diabaikan, beserta *tile* yang mengikutinya. Posisi indeks akan bergeser ke karakter pertama yang belum tertandai yang letaknya setelah *tile* tersebut. Apabila jarak indeks saat itu ke *tile* berikutnya lebih besar dari *search-length* maka token dari TextStringT dapat dibentuk berdasarkan panjang *search-length (s)*.

Akan tetapi jika setiap karakter pada TextStringT belum tertandai, maka pada baris kode ke-66 sampai dengan 78, token dengan panjang (*s*) dapat dibentuk sampai dengan $T[_{0..n-1}]$ kemudian hash value dari setiap token tersebut dihitung (baris kode ke-75).

Pada baris kode ke-84, jika karakter pada TextStringP telah ada yang tertandai, maka cek jarak posisi indeks yang ada pada saat itu ke *tile* berikutnya, apabila lebih kecil dari *search-length (s)*, maka bagian yang belum tertandai pada TextStringP mulai dari posisi indeks tersebut akan diabaikan, beserta *tile* yang mengikutinya. Posisi indeks akan bergeser ke karakter pertama yang belum tertandai yang letaknya setelah *tile* tersebut. Apabila jarak indeks saat itu ke *tile* berikutnya lebih besar dari *search-length (s)* maka token dari TextStringP dapat dibentuk berdasarkan panjang *search-length (s)* dan pada baris kode ke-102 langsung dihitung hash value dari token TextStringP tersebut, untuk selanjutnya pada baris kode ke-113 dilakukan pengecekan ke HashTable dari TextStringT.

Baris kode ke-115 sampai dengan 156 merupakan implementasi dari *pseudocode* pada fase *scanpattern* dibaris *pseudocode* ke-13 sampai dengan 21. *Code Snippets function scanpattern* selengkapnya di gambarkan pada Gambar 5.21.

```
1   Function ScanPattern(ByRef S As Integer, ByVal TextStringP As String, ByVal
    TextStringT As String)
2   '+++++++++++++++ Create KarpRabin HashTable Sepanjang (T[t..t+s-1]) ++++++++
3   Dim increment As Integer = 1
4   Dim x As Integer
5   Dim xTokenAscii As Integer
6   Dim vxSum As Integer
7   Dim xIndexArrT As Integer
8   Dim xIndexArrP As Integer
9   Dim K As Integer
10  Dim ArrSubStringT As String
11  Dim ArrSubStringP As String
12  Dim n As Integer
13  Dim A1 As Byte
14  Dim An As Byte
15  Dim Zz As Byte = 36
16
17  Call xReleaseArray()
18
19  If xKondisiMark = True Then
20     For i = 1 To UBound(xStatusMark_T)
21         If xStatusMark_T(i - 1).xStatus = 0 Then
22            vxSum = vxSum + 1
23         Else
24            If vxSum <> 0 Then
25              If vxSum < S Then
26                'Release vxSum
```

```
27                'Maju ke Unmark string yg ada didepan posisi pointer saat ini
28                 vxSum = 0
29              Else
30                 For z = (i - vxSum) To (i - S)
31                    ReDim Preserve ArrTokenString(xIndexArrT)
32                    ArrTokenString(xIndexArrT).xTokenStringT =
                                          Mid(TextStringT.ToLower, z, S)
33                    ArrTokenString(xIndexArrT).xIndex = z
                      xIndexArrT = xIndexArrT + 1
34                 Next
35                 vxSum = 0
36              End If
37           End If
38        End If
39
40        If i = UBound(xStatusMark_T) And vxSum <> 0 Then
41           For z = (i - vxSum) To (i - S)
42               ReDim Preserve ArrTokenString(xIndexArrT)
43               ArrTokenString(xIndexArrT).xTokenStringT =
                                      Mid(TextStringT.ToLower, z + 1, S)
44               ArrTokenString(xIndexArrT).xIndex = z + 1
45               xIndexArrT = xIndexArrT + 1
46           Next
47           vxSum = 0
48        End If
49     Next
50
51        If Not ArrTokenString Is Nothing Then
52           For x = 0 To (xIndexArrT - 1)
53               xTokenAscii = 0
54               ArrSubStringT = ArrTokenString(x).xTokenStringT.ToLower
55               n = Len(ArrTokenString(x).xTokenStringT)
56               A1 = Asc(Microsoft.VisualBasic.Left(ArrSubStringT, 1))
57               An = Asc(Microsoft.VisualBasic.Right(ArrSubStringT, 1))
58               xTokenAscii = A1 + An + (n * Zz)
59               ArrTokenString(x).xHashValue = xTokenAscii
60           Next
61        Else
62           Return S
63        End If
64  Else
65      'Jika semua Element di String T belum di Mark
66      For x = 0 To Len(TextStringT) - S
67          xTokenAscii = 0
68          ReDim Preserve ArrTokenString(x)
69          ArrTokenString(x).xTokenStringT =
70                                    Mid(TextStringT.ToLower, increment, S)
71          ArrSubStringT = ArrTokenString(x).xTokenStringT.ToLower
72          n = Len(ArrTokenString(x).xTokenStringT)
73          A1 = Asc(Microsoft.VisualBasic.Left(ArrSubStringT, 1))
74          An = Asc(Microsoft.VisualBasic.Right(ArrSubStringT, 1))
75          xTokenAscii = A1 + An + (n * Zz)
76          ArrTokenString(x).xHashValue = xTokenAscii
77          increment = increment + 1
78      Next
79  End If
80  vxSum = 0
81  '++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
82  '+++++++++++++++++ Cek Kondisi Mark pada String P ++++++++++++++++++++++
83  '++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
84  If xKondisiMark = True Then
85     For i = 1 To UBound(xStatusMark_P)
86         If xStatusMark_P(i).xStatus = 0 Then
87            vxSum = vxSum + 1
```

```
88              If i = UBound(xStatusMark_P) Then
89              Dim xCurrent_IndexP As Integer
90                 For z = (i - (vxSum - 1)) To (i - S)
91                     ReDim Preserve ArrTokenPattern(xIndexArrP)
92                     ArrTokenPattern(xIndexArrP).xTokenStringP =
                                        Mid(TextStringP.ToLower, z + 1, S)
93                     ArrTokenPattern(xIndexArrP).xIndex = (z + 1)
94                     xCurrent_IndexP = xIndexArrP
95                     xIndexArrP = xIndexArrP + 1
96                     vxSum = 0
97                     xTokenAscii = 0
98                     ArrSubStringP =
                            ArrTokenPattern(xCurrent_IndexP).xTokenStringP.ToLower
99                     n = Len(ArrTokenPattern(xCurrent_IndexP).xTokenStringP)
100                    A1 = Asc(Microsoft.VisualBasic.Left(ArrSubStringP, 1))
101                    An = Asc(Microsoft.VisualBasic.Right(ArrSubStringP, 1))
102                    xTokenAscii = A1 + An + (n * Zz)
103                    ArrTokenPattern(xCurrent_IndexP).xHashValue = xTokenAscii
104
105                    '++++++++ Cari HashValue_P kedalam HashTable(T) ++++++++++
106                    Dim ii As Integer
107                    Dim P_Pointer As Integer
108                    Dim T_Pointer As Integer
109                    Dim xPosisiPointer_P As Integer
110                    Dim xPosisiPointer_T As Integer
111
112                    For ii = 0 To (ArrTokenString.GetLength(0) - 1)
113                        If ArrTokenString(ii).xHashValue =
                                ArrTokenPattern(xCurrent_IndexP).xHashValue Then
114                           K = S
115                           xStringP = TextStringP.ToLower
116                           xStringT = TextStringT.ToLower
117                           P_Pointer = ArrTokenPattern(xCurrent_IndexP).xIndex
118                           T_Pointer = ArrTokenString(ii).xIndex
119                           xPosisiPointer_P = P_Pointer
120                           xPosisiPointer_T = T_Pointer
121
122                          If (P_Pointer + S) <= Len(xStringP) Or
                                (T_Pointer + S) <= Len(xStringT) Then
123                             Do While Mid(xStringP, P_Pointer + S, 1) =
                                   Mid(xStringT, T_Pointer + S, 1) And
                                   xStatusMark_P((P_Pointer-1)+S).xStatus = 0
                                   And xStatusMark_T((T_Pointer-1)+S).xStatus = 0
124                                K = K + 1
125                                P_Pointer = P_Pointer + 1
126                                T_Pointer = T_Pointer + 1
127                                If (P_Pointer + 1) > Len(xStringP) Or
                                      (T_Pointer + S) > Len(xStringT) Then
128                                   Exit Do
129                                End If
130                             Loop
131                         End If
132                         If K > (2 * S) Then
133                            LMax = K
134                            Exit Function
135                         Else
136                            LMax = K
137                            'add match(p,t,k) to match_list
138                            Dim xVal As Integer
139                            If ListOfMaxMatch Is Nothing Then
140                               xVal = 0
141                            Else
142                               xVal = UBound(ListOfMaxMatch) + 1
143                            End If
```

```
144
145                             If ListOfMaxMatch Is Nothing Then
146                                ReDim Preserve ListOfMaxMatch(xVal)
147                                ListOfMaxMatch(xVal).arrMatchPTK =
                                   (xPosisiPointer_P) & "," & (xPosisiPointer_T)
                                   & "," & (K)
148                             Else
149                                ReDim Preserve ListOfMaxMatch(xVal)
150                                ListOfMaxMatch(xVal).arrMatchPTK =
                                   (xPosisiPointer_P) & "," & (xPosisiPointer_T)
                                   & "," & (K)
151                             End If
152                          End If
153                       End If
154                    Next
155                 Next
156              End If
157           Else
158              If vxSum <> 0 Then
159                 If vxSum < S Then
160                   'Release vxSum
161                   'Maju ke Unmark string yg ada didepan posisi pointer saat ini
162                    vxSum = 0
163                 Else
164                    Dim xCurrent_IndexP As Integer
165                    For z = (i - vxSum) To (i - S)
166                        ReDim Preserve ArrTokenPattern(xIndexArrP)
167                        ArrTokenPattern(xIndexArrP).xTokenStringP =
                                              Mid(TextStringP.ToLower, z, S)
168                        xCurrent_IndexP = xIndexArrP
169                        xIndexArrP = xIndexArrP + 1
170                    Next
171                    vxSum = 0
172                    xTokenAscii = 0
173
174                    ArrSubStringP =
                             ArrTokenPattern(xCurrent_IndexP).xTokenStringP.ToLower
175                    n = Len(ArrTokenPattern(xCurrent_IndexP).xTokenStringP)
176                    A1 = Asc(Microsoft.VisualBasic.Left(ArrSubStringP, 1))
177                    An = Asc(Microsoft.VisualBasic.Right(ArrSubStringP, 1))
178                    xTokenAscii = A1 + An + (n * Zz)
179                    ArrTokenPattern(xCurrent_IndexP).xHashValue = xTokenAscii
180
181                    '++++++++ Cari HashValue_P kedalam HashTable(T) ++++++++++
182                    Dim ii As Integer
183                    Dim P_Pointer As Integer
184                    Dim T_Pointer As Integer
185
186                    For ii = 0 To (ArrTokenString.GetLength(0) - 1)
187                        If ArrTokenString(ii).xHashValue =
                              ArrTokenPattern(xCurrent_IndexP).xHashValue Then
188                           K = S
189                           xStringP = TextStringP.ToLower
190                           xStringT = TextStringT.ToLower
191                           P_Pointer = xStringP.IndexOf
                                   (ArrTokenPattern(xCurrent_IndexP).xTokenStringP)
192                           T_Pointer = xStringT.IndexOf
                                      (ArrTokenString(ii).xTokenStringT)
193
194                           If (P_Pointer + S) < Len(xStringP) And
195                              (T_Pointer + S) < Len(xStringT) Then
196                              Do While xStringP.Chars(P_Pointer + S) =
                                         xStringT.Chars(T_Pointer + S) And
                                         xStatusMark_P(P_Pointer + S).xStatus = 0
```

```
                                    And xStatusMark_T(T_Pointer+S).xStatus = 0
197                             K = K + 1
198                             P_Pointer = P_Pointer + 1
199                             T_Pointer = T_Pointer + 1
200
201                             If (P_Pointer) > Len(xStringP) Or
                                   (T_Pointer) > Len(xStringT) Then
202                                Exit Do
203                             End If
204                          Loop
205                       End If
206
207                       If K > (2 * S) Then
208                          LMax = K
209                          Exit Function
210                       Else
211                          LMax = K
212                          'add match(p,t,k) to match_list
213                          Dim xVal As Integer
214                          If ListOfMaxMatch Is Nothing Then
215                             xVal = 0
216                          Else
217                             xVal = UBound(ListOfMaxMatch) + 1
218                          End If
219
220                          If ListOfMaxMatch Is Nothing Then
221                             ReDim Preserve ListOfMaxMatch(xVal)
222                             ListOfMaxMatch(xVal).arrMatchPTK =
                                               (P_Pointer + 1) & "," &
                                               (T_Pointer + 1) & "," & (K)
223                          Else
224                              For z = 0 To UBound(ListOfMaxMatch)
225                                 If ListOfMaxMatch(z).arrMatchPTK =
                                       (P_Pointer + 1) & "," & (T_Pointer + 1)
                                       & "," & (K) Then
226                                        Exit For
227                                 End If
228                              Next
229                          End If
230                       End If
231                    End If
232                 Next
233              End If
234           End If
235        End If
236     Next
237  Else
238     'Jika Semua Element di String P belum di Mark++++++++++++++++++++++
239     '++++++++ Create HashValue untuk Token String P Sepanjang (s) ++++++++
240     Dim xIndexP As Integer
241     For xIndexP = 0 To (Len(TextStringP) - S)
242         xTokenAscii = 0
243         ReDim Preserve ArrTokenPattern(xIndexP)
244         ArrTokenPattern(xIndexP).xTokenStringP = Mid(TextStringP.ToLower,
                                                     xIndexP + 1, S)
245         ArrSubStringP = ArrTokenPattern(xIndexP).xTokenStringP.ToLower
246         n = Len(ArrTokenPattern(xIndexP).xTokenStringP)
247         A1 = Asc(Microsoft.VisualBasic.Left(ArrSubStringP, 1))
248         An = Asc(Microsoft.VisualBasic.Right(ArrSubStringP, 1))
249         xTokenAscii = A1 + An + (n * Zz)
250         ArrTokenPattern(xIndexP).xHashValue = xTokenAscii
251
252         '+++++++++ Cari HashValue_P kedalam HashTable(T) +++++++++++++++
253         Dim ii As Integer
```

```
254             Dim P_Pointer As Integer
255             Dim T_Pointer As Integer
256
257             For ii = 0 To (ArrTokenString.GetLength(0) - 1)
258                 If ArrTokenString(ii).xHashValue =
                       ArrTokenPattern(xIndexP).xHashValue Then
259                    K = S
260                    P_Pointer = ((xIndexP + 1) + S)
261                    T_Pointer = (ii + 1) + S
262                    xStringP = TextStringP.ToLower
263                    xStringT = TextStringT.ToLower
264
265                    If (P_Pointer) <= Len(xStringP) Or
266                       (T_Pointer) <= Len(xStringT) Then
267                       Do While Mid(xStringP, P_Pointer, 1) =
                                                    Mid(xStringT, T_Pointer, 1)
268                          K = K + 1
269                          P_Pointer = P_Pointer + 1
270                          T_Pointer = T_Pointer + 1
271
272                          If (P_Pointer) > Len(xStringP) Or
                                (T_Pointer) > Len(xStringT) Then
273                                Exit Do
274                          End If
275                       Loop
276                    End If
277
278                    If K > (2 * S) Then
279                       LMax = K
280                       Exit Function
281                    Else
282                       LMax = K
283                       'add match(p,t,k) to match_list
284                       Dim xVal As Integer
285                       If ListOfMaxMatch Is Nothing Then
286                          xVal = 0
287                       Else
288                          xVal = UBound(ListOfMaxMatch) + 1
289                       End If
290                       ReDim Preserve ListOfMaxMatch(xVal)
291                       ListOfMaxMatch(xVal).arrMatchPTK = (xIndexP + 1) & "," &
                                                            (ii + 1) & "," & (K)
292                    End If
293                End If
294             Next
295         Next
296 End If
297 End Function
```

Gambar 5.21 Code snippets proses scanpattern

5.9.3 Fase *markstring*

Fase *markstring* adalah fase penandaan string yang terdapat pada *list-of-maximal-match* hasil *scanpattern*. *List-of-maximal-match* menyimpan posisi *index* dari setiap TextStringP dan TextStringT yang memiliki kesamaan sejumlah s. Pada fase *scanpattern* string dinyatakan sama jika *hash value* nya sama, dan dalam fase *markstring* pada baris kode ke-28 sebelum dilakukan penandaan terhadap string yang terdapat pada *list-of-maximal-match* akan dilakukan

pengecekan kembali terhadap setiap karakter yang ada, dan harus dipastikan pula dalam proses penandaan string tersebut, string dalam keadaan belum tertandai, karena jika suatu string telah tertandai maka string tersebut mestilah telah bagian dari *tile* lainnya. Implementasi fase *markstring* digambarkan pada Gambar 5.22.

```
1   Function Markstring(ByRef s As Integer)
2   Try
3      Dim oHash As New Hashtable
4      Dim p As String
5      Dim t As Short
6      Dim k As Short
7      If ListOfMaxMatch Is Nothing Then
8         s = 0
9      Else
10        For jum As Byte = 0 To UBound(ListOfMaxMatch)
11            p = Split(ListOfMaxMatch(jum).arrMatchPTK, ",")(0)
12            t = Split(ListOfMaxMatch(jum).arrMatchPTK, ",")(1)
13            k = Split(ListOfMaxMatch(jum).arrMatchPTK, ",")(2)
14            oHash.Add((p) & "," & (t), k)
15        Next
16        Dim xJum As Byte = oHash.Count
17        Dim oDataView As DataView = SortHashtable(oHash)
18        For iRow As Long = 0 To oDataView.Count - 1
19            Dim sKey As String = oDataView(iRow)("key")
20            Dim sValue As String = oDataView(iRow)("value")
21            Dim xPosisiPointer_P As Short = Split(sKey, ",")(0)
22            Dim xPosisiPointer_T As Short = Split(sKey, ",")(1)
23           '++++++++++++++ Markstring P ++++++++++++++++++
24            Dim i As Short
25            Dim ii As Short
26            Dim xSum As Short
27            For i = 0 To (sValue - 1)
28                If Mid(xStringP, xPosisiPointer_P + i, 1) = Mid(xStringT,
                     xPosisiPointer_T + i, 1) And
                     xStatusMark_P(xPosisiPointer_P + i).xStatus = 0 And
                     xStatusMark_T(xPosisiPointer_T + i).xStatus = 0 Then
29                    xSum = xSum + 1
30                    If xSum = sValue Then
31                      'Marks String P
32                      For ii = xPosisiPointer_P To ((xPosisiPointer_P+sValue)-1)
33                          xStatusMark_P(ii).xStatus = 1
34                      Next
35                     'Marks String T
36                      For ii = xPosisiPointer_T To ((xPosisiPointer_T+sValue)-1)
37                          xStatusMark_T(ii).xStatus = 1
38                      Next
39                      Length_Of_Token_Tile = Length_Of_Token_Tile + sValue
40                      xJumlahMark = Length_Of_Token_Tile
41                    End If
42                ElseIf (sValue - (sValue - xSum)) >= s Then
43                    'Tambah unmark token ke dalam List_Of_MaxMatch
44                    ReDim Preserve ListOfMaxMatch(UBound(ListOfMaxMatch))
45                    ListOfMaxMatch(UBound(ListOfMaxMatch)).arrMatchPTK =
                      xPosisiPointer_T + i & "," & xPosisiPointer_P + i & "," &
46                    (sValue - xSum)
47                End If
48                oHash.Remove(sKey)
49            Next
50            xSum = 0
51        Next
```

```
52          xKondisiMark = True
53          Dim xTot As Byte = oHash.Count
54          Erase ListOfMaxMatch
55       End If
56    Catch ex As Exception
57          MsgBox(ex.Message, MsgBoxStyle.OkOnly, "Error Fase Markstring")
58    End Try
59    End Function
```

Gambar 5.22 Code snippets proses markstring

5.10 Dice Coefficient

Pada akhir dari proses pengecekan didapatkan jumlah procedure yang memiliki kemiripan, persentase kemiripan dari kedua modul *source code* dapat dihitung dengan metode dice coefficient yang digambarkan pada Gambar 5.23.

```
1   xSimilarityValue = ((2 * Val(listDataSource.Count)) /
                        (Val(SumProcedureT1.Text) + Val(SumProcedureT2.Text))*100)
```

Gambar 5.23 Code snippets perhitungan persentase kemiripan

BAB VI
HASIL PENELITIAN DAN PEMBAHASAN

6.1 Pengujian Modul Source Code

Pada *modul source code* sebelum dilakukan pengujian terlebih dahulu telah dilakukan beberapa modifikasi, diantaranya mengubah urutan *statement* yang ada pada *procedure* atau *function*, mengubah *statement* yang ada pada procedure atau function, mengubah nama *procedure* atau *function*, mengubah nama variable, mengubah type data variable dengan type data lain yang sejenis, mengubah atau menambah komentar, mengubah pemanggilan suatu *procedure* dengan mengganti nama *procedure* yang dipanggil dan akan dilakukan juga uji coba terdapat modul hasil *copy-paste.*

Pada sub bab 6.1.1 akan dilakukan pengujian dengan mengubah urutan statement, pada sub bab 6.1.2 akan dilakukan pengujian dengan perubahan statement yang ada didalam procedure, sub bab 6.1.3 akan dilakukan pengujian terhadap procedure yang sebagian *syntax*-nya berasal dari procedure pembanding.

Untuk hasil pengujian dari beberapa bahasa pemrograman yang menjadi target deteksi, dapat dilihat pada sub bab 6.1.4, 6.1.5, 6.1.6 dan 6.1.7. Pada proses pengujian yang dilakukan, peneliti melakukan serangkaian uji coba menggunakan seting parameter sensitifitas deteksi 100% dikarenakan data yang diuji sebenarnya memiliki kesamaan, hanya saja telah mengalami modifikasi, diantaranya mengubah nama variable beserta type data, mengubah nama procedure, serta penambahan komentar dan perubahan nama procedure yang dipanggil didalam body. Dengan seting deteksi filtering variable dan non filtering variable diharapkan sistem mampu memberikan output yang sesuai dengan data yang diuji.

Persentase similaritas yang dihasilkan oleh sistem dihitung berdasarkan rumus *dice coefficient* similarity ((*n**2)/(t1+t2)*100), dimana *n* merupakan jumlah procedure hasil pembandingan yang dianggap sama oleh sistem dan t1,t2 merupakan jumlah masing-masing procedure yang terdapat pada *source code* pengujian. Suatu procedure dianggap memiliki kesamaan jika jumlah string yang

tertandai pada procedure yang diuji mencapai lebih besar atau sama dengan nilai sensitifitas deteksi yang telah ditentukan saat pengujian.

6.1.1 Pengujian dengan mengubah urutan *statement*

Pada pengujian ini, akan dilakukan perubahan urutan *statement* yang terdapat pada procedure. Gambar 6.1 menggambarkan keadaan procedure sebelum dilakukan perubahan urutan *statement*.

```
1   Private Sub Simpan()
2   If kdjurusan.Text = "" Then
3      kdjurusan.SetFocus
4   ElseIf nmjurusan.Text = "" Then
5      nmjurusan.SetFocus
6   Else
7      Set rsdata = New ADODB.Recordset
8      strsql = "Insert into jurusan values('" & kdjurusan.Text & "',
9               '" & nmjurusan.Text & "')"
10     rsdata.Open strsql, conn, adOpenDynamic, adLockOptimistic
11  MsgBox "Data telah tersimpan", vbOKOnly + vbInformation, "INFO"
12  Frame1.Enabled = False
13  Call TampilkanDataGrid
14  Call Bersihkan
15  Call TutupKunci
16  End If
17  End Sub
```

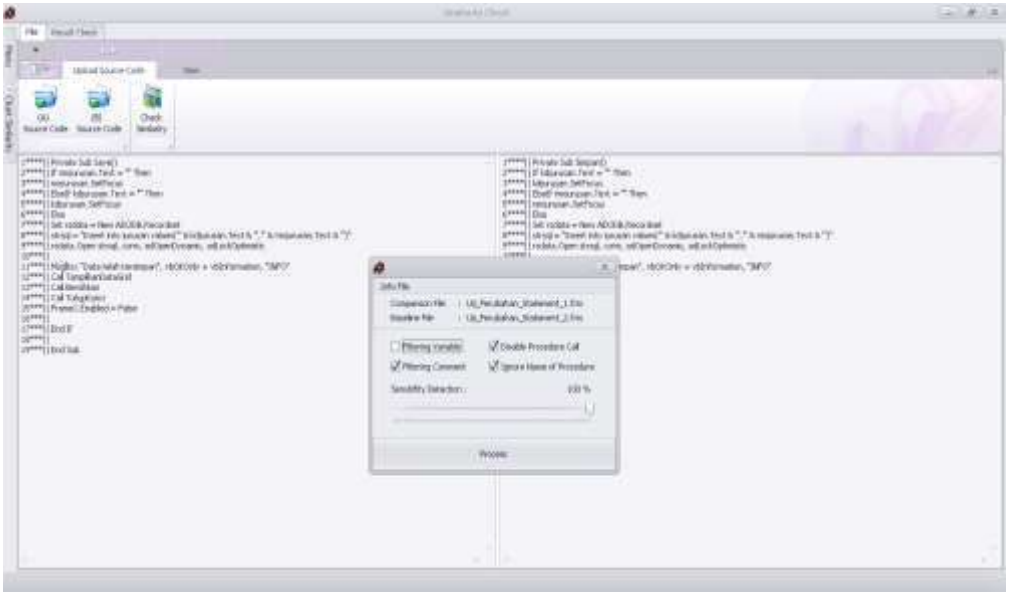
Gambar 6.1 Procedure sebelum dilakukan perubahan urutan statement

Modifikasi mengubah urutan *statement* beserta nama *procedure* yang telah dilakukan ditunjukkan pada Gambar 6.2.

```
1   Private Sub Save()
2   If nmjurusan.Text = "" Then
3      nmjurusan.SetFocus
4   ElseIf kdjurusan.Text = "" Then
5      kdjurusan.SetFocus
6   Else
7      Set rsdata = New ADODB.Recordset
8      strsql = "Insert into jurusan values('" & kdjurusan.Text & "',
9               '" & nmjurusan.Text & "')"
10     rsdata.Open strsql, conn, adOpenDynamic, adLockOptimistic
11  MsgBox "Data telah tersimpan", vbOKOnly + vbInformation, "INFO"
12  Call Bersihkan
13  Call TampilkanDataGrid
14  Call TutupKunci
15  Frame1.Enabled = False
16  End If
17  End Sub
```

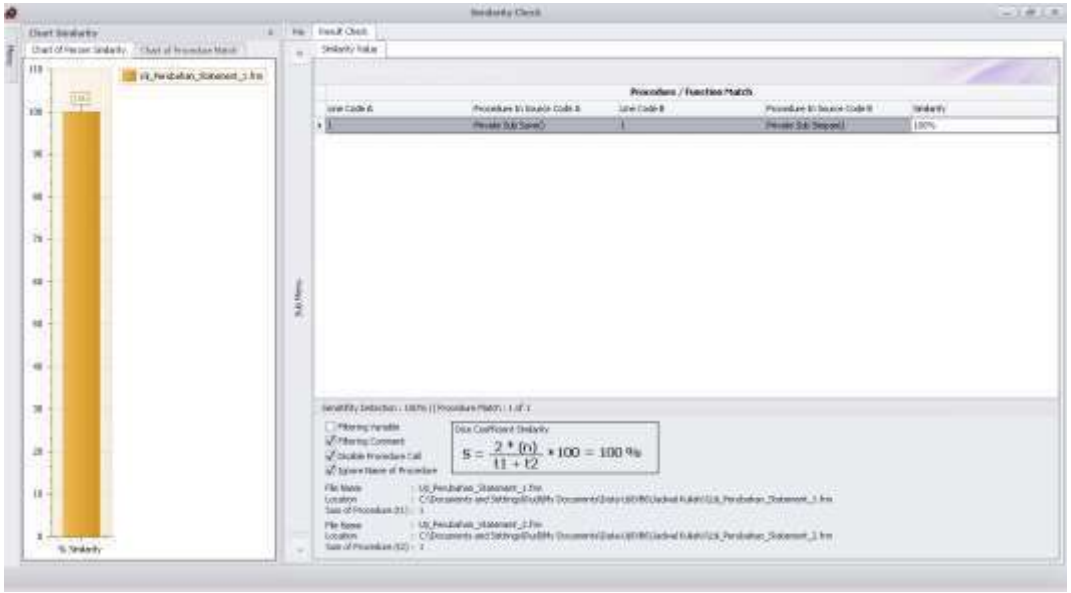
Gambar 6.2 Procedure setelah dilakukan perubahan urutan statement

Dilakukan pengujian pada kedua procedure yang ada pada Gambar 6.1 dan 6.2, menggunakan parameter sensitifitas deteksi 100% seperti yang terlihat pada Gambar 6.3.



Gambar 6.3 Parameter deteksi

Hasil pengujian pada Gambar 6.4, menyatakan kedua procedure tersebut memiliki kesamaan 100%, ini membuktikan algoritma Running Karp-Rabin Greedy String Tiling yang diterapkan pada sistem mampu menemukan perubahan posisi *statement* yang telah dilakukan.



Gambar 6.4 Hasil pengujian terhadap perubahan urutan statement

6.1.2 Pengujian dengan perubahan *statement*

Pada pengujian ini dilakukan sedikit perubahan *statement* yang terdapat didalam procedure, kemudian akan dilakukan pengujian dengan parameter sensitifitas deteksi 100% dan 75%, untuk melihat perbedaan hasil deteksi yang dihasilkan oleh sistem. Gambar 6.5 menggambarkan keadaan procedure sebelum dilakukan perubahan statement.

```
1   Private Sub Simpan()
2   If kdjurusan.Text = "" Then
3     kdjurusan.SetFocus
4   ElseIf nmjurusan.Text = "" Then
5     nmjurusan.SetFocus
6   Else
7     Set rsdata = New ADODB.Recordset
8     strsql = "Insert into jurusan values('" & kdjurusan.Text & "',
9               '" & nmjurusan.Text & "')"
10    rsdata.Open strsql, conn, adOpenDynamic, adLockOptimistic
11
12    MsgBox "Data telah tersimpan", vbOKOnly + vbInformation, "INFO"
13    Frame1.Enabled = False
14    Call TampilkanDataGrid
15    Call Bersihkan
16    Call TutupKunci
17  End If
18  End Sub
```

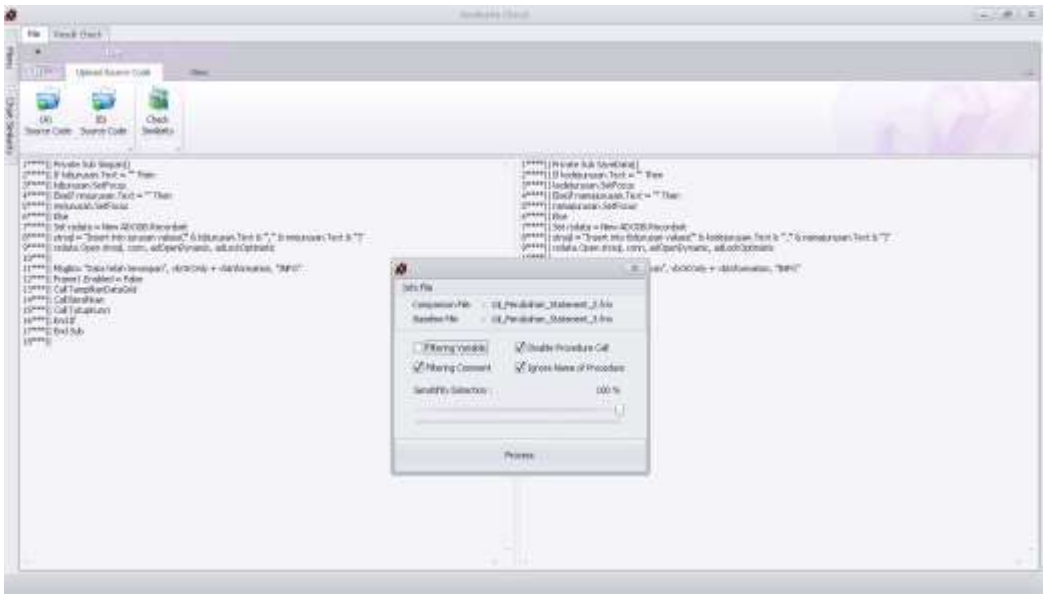
Gambar 6.5 Procedure sebelum dilakukan perubahan statement

Modifikasi perubahan statement beserta nama procedure dilakukan seperti pada Gambar 6.6.

```
1   Private Sub SimpanData()
2   If kodejurusan.Text = "" Then
3     kodejurusan.SetFocus
4   ElseIf namajurusan.Text = "" Then
5     namajurusan.SetFocus
6   Else
7     Set rsdata = New ADODB.Recordset
8     strsql = "Insert into tbljurusan values('" & kodejurusan.Text &
9               "','" & namajurusan.Text & "')"
10    rsdata.Open strsql, conn, adOpenDynamic, adLockOptimistic
11    MsgBox "Data telah tersimpan", vbOKOnly + vbInformation, "INFO"
12    Call TampilkanDataGrid
13    Call Bersihkan
14    Frame1.Enabled = False
15    Call TutupKunci
16  End If
17  End Sub
```

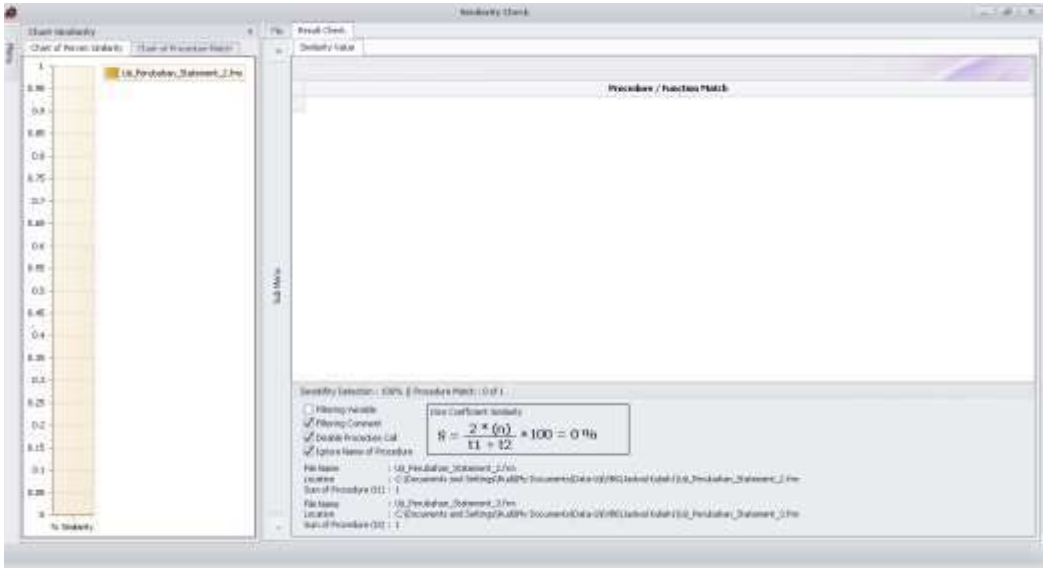
Gambar 6.6 Procedure setelah dilakukan perubahan statement

Pada pengujian pertama dengan parameter sensitifitas deteksi 100%, seperti yang terlihat pada Gambar 6.7.



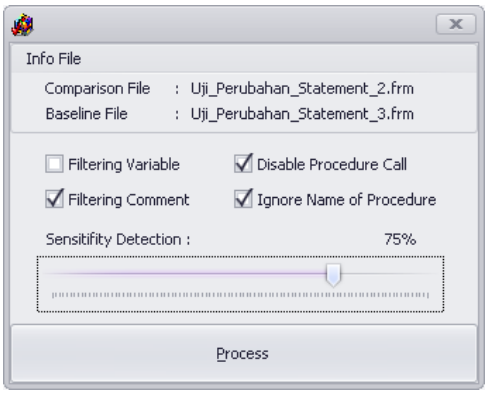
Gambar 6.7 Parameter pengujian pertama

Hasil pengujian pada Gambar 6.8, menunjukkan tidak ditemukan adanya kesamaan antara kedua procedure, mengingat telah dilakukan perubahan statement maka output yang dihasilkan berkesesuaian dengan data procedure yang diuji serta parameter pengujian yang digunakan.



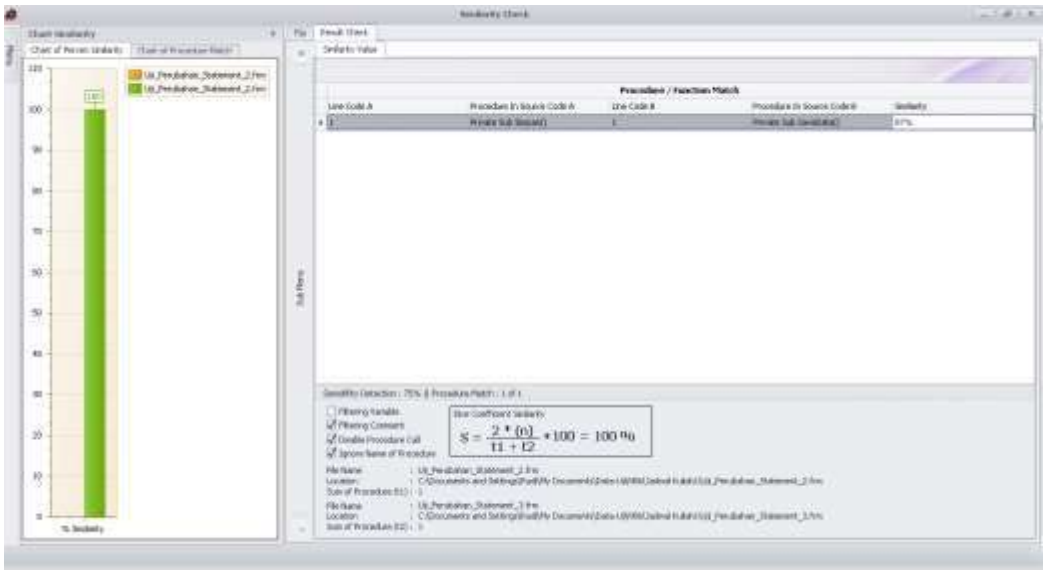
Gambar 6.8 Hasil pengujian pertama terhadap perubahan statement

Pada data yang sama dilakukan pengujian kedua dengan nilai sensitifitas deteksi 75% seperti yang terlihat pada Gambar 6.9.



Gambar 6.9 Parameter pengujian kedua terhadap perubahan statement

Hasil pengujian pada Gambar 6.10, menunjukkan ditemukan adanya kesamaan antara kedua procedure, similaritas yang didapat oleh sistem pada procedure tersebut sebesar 87% yang artinya nilai similaritas yang dihasilkan lebih besar dari nilai sensiitifitas pengujian yang ditentukan saat pengujian yaitu sebesar 75%, sehingga output yang dihasilkan menyatakan source code tersebut memiliki kesamaan 100%, berkesesuaian dengan data dan parameter pengujian yang digunakan.



Gambar 6.10 Hasil pengujian kedua terhadap perubahan statement

6.1.3 Pengujian dengan mengambil sebagian isi dari procedure

Gambar 6.11, menunjukkan keadaan procedure dalam kondisi lengkap dengan *statement* validasi input, procedure ini akan digunakan sebagai procedure pembandingnya.

```
1   Private Sub SaveData()
2   If kodejurusan.Text = "" Then
3     kodejurusan.SetFocus
4   ElseIf namajurusan.Text = "" Then
5     namajurusan.SetFocus
6   Else
7     Set rsdata = New ADODB.Recordset
8     strsql = "Insert into tbljurusan values('" & kodejurusan.Text &
9              "','" & namajurusan.Text & "')"
10    rsdata.Open strsql, conn, adOpenDynamic, adLockOptimistic
11
12    MsgBox "Data telah tersimpan", vbOKOnly + vbInformation, "INFO"
13    Call TampilkanDataGrid
14    Call Bersihkan
15    Frame1.Enabled = False
16    Call TutupKunci
17  End If
18  End Sub
```

Gambar 6.11 Procedure dalam kondisi lengkap

Pada Gambar 6.12, telah dilakukan pengambilan sebagian isi dari procedure yang ada pada Gambar 6.11. Procedure yang ada pada Gambar 6.12 akan diberlakukan sebagai procedure yang diuji.

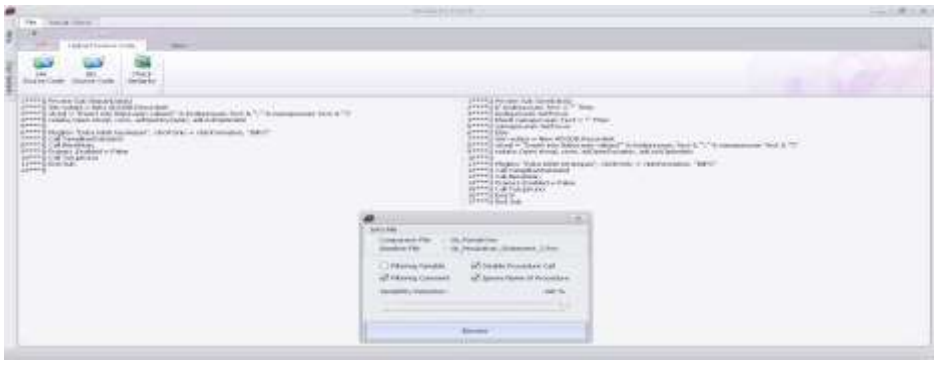
```
1   Private Sub SimpanData()
2     Set rsdata = New ADODB.Recordset
3     strsql = "Insert into tbljurusan values('" & kodejurusan.Text &
4              "','" & namajurusan.Text & "')"
5     rsdata.Open strsql, conn, adOpenDynamic, adLockOptimistic
6
7     MsgBox "Data telah tersimpan", vbOKOnly + vbInformation, "INFO"
8     Call TampilkanDataGrid
9     Call Bersihkan
10    Frame1.Enabled = False
11    Call TutupKunci
12  End Sub
```

Gambar 6.12 Procedure hasil pengambilan sebagian

Pada 2 procedure diatas akan dilakukan 3x pengujian. Dimana pengujian pertama akan dilakukan dengan sensitifitas deteksi 100%. Kemudian pada pengujian kedua dan ketiga procedure yang diuji akan ditukar sebagai procedure

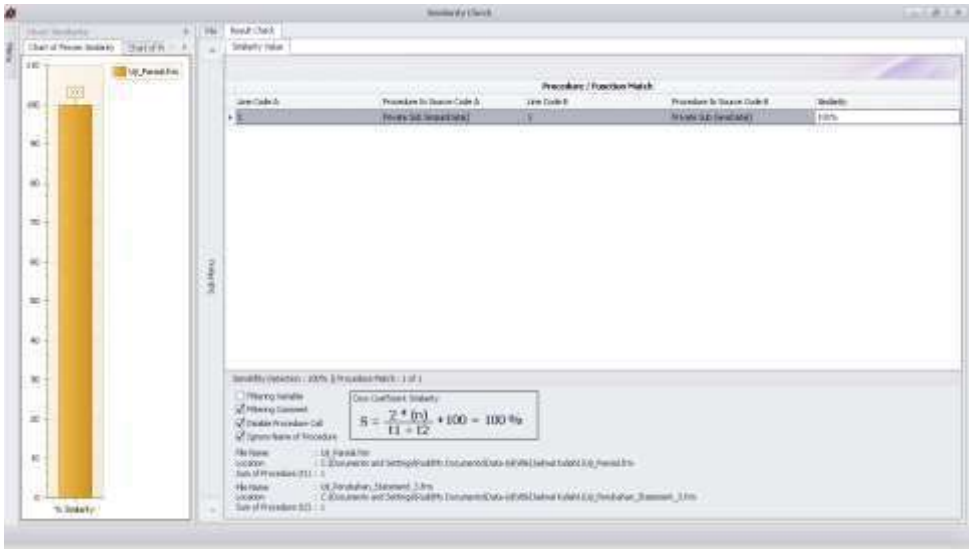
pembandingannya dengan sensitifitas deteksi yang akan digunakan 100% dan 50%. Selengkapnya mengenai pengujian akan dibahas berikut ini.

1. Pengujian pertama menggunakan sensitifitas deteksi sebesar 100% seperti yang terlihat pada Gambar 6.13. Dimana procedure yang ada pada Gambar 6.12 menjadi procedure yang akan diuji kemiripannya dengan procedure pada Gambar 6.11 sebagai pembandingnya.



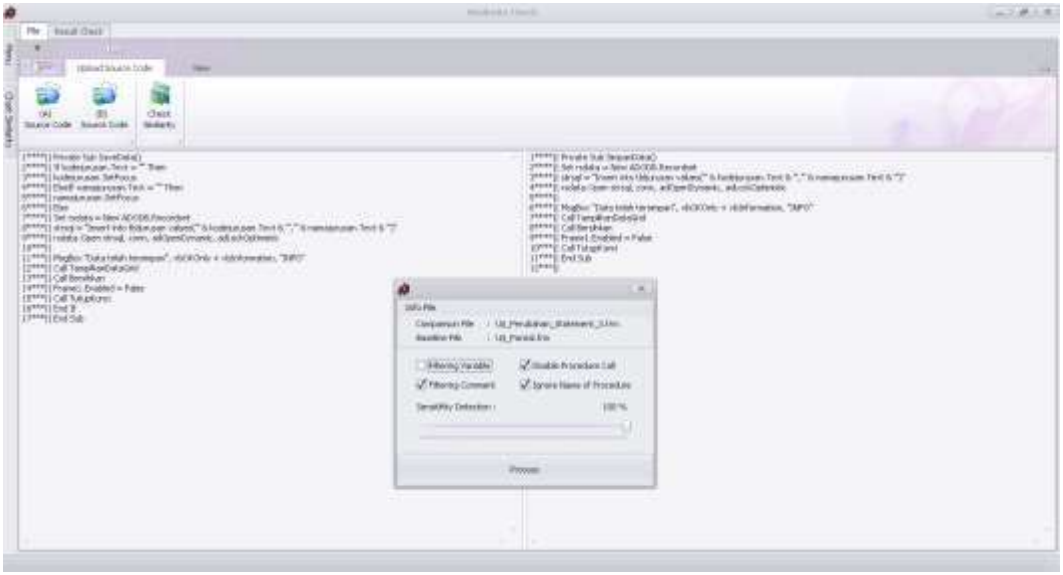
Gambar 6.13 Parameter pengujian pertama

Dari hasil pengujian pada Gambar 6.14, didapat kemiripan antara 2 procedure yang ada sebesar 100%, ini menandakan keseluruhan *syntax* yang ada pada procedure yang diuji (Gambar 6.12) juga terdapat/memiliki kesamaan pada procedure pembandingnya (Gambar 6.11).



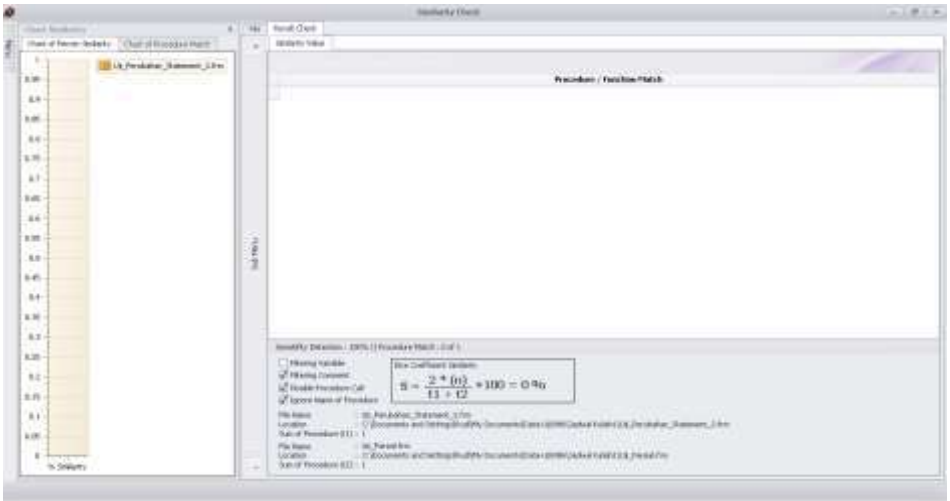
Gambar 6.14 Hasil pengujian pertama

2. Pengujian kedua dilakukan dengan tetap menggunakan sensitifitas deteksi sebesar 100%, akan tetapi procedure yang semula diuji ditukar menjadi procedure pembandingnya dan yang semula procedure pembanding menjadi procedure yang diuji. Gambar 6.15 merupakan parameter pengujian kedua.



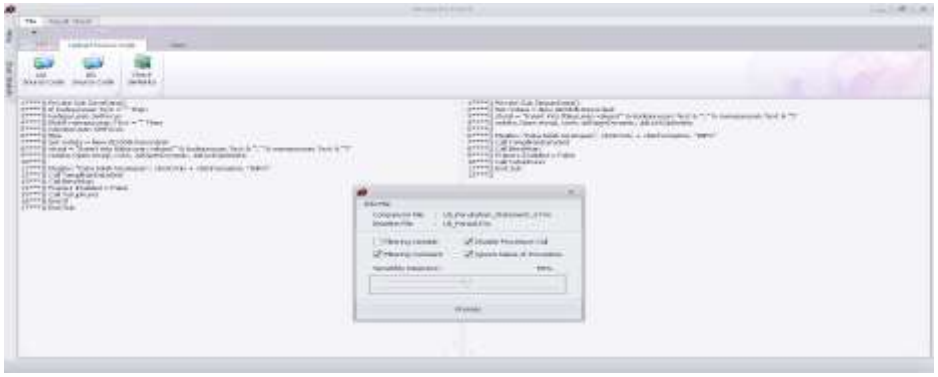
Gambar 6.15 Parameter pengujian kedua

Dari hasil pengujian kedua pada Gambar 6.16, dengan parameter sensitifitas deteksi 100%, output sistem tidak menemukan adanya kesamaan dari 2 procedure yang dibandingkan, ini berarti tidak 100% semua baris *syntax* yang ada pada procedure yang diuji memiliki kesamaan dengan procedure pembandingnya. Output yang dihasilkan sistem berkesesuaian dengan data yang diujikan.



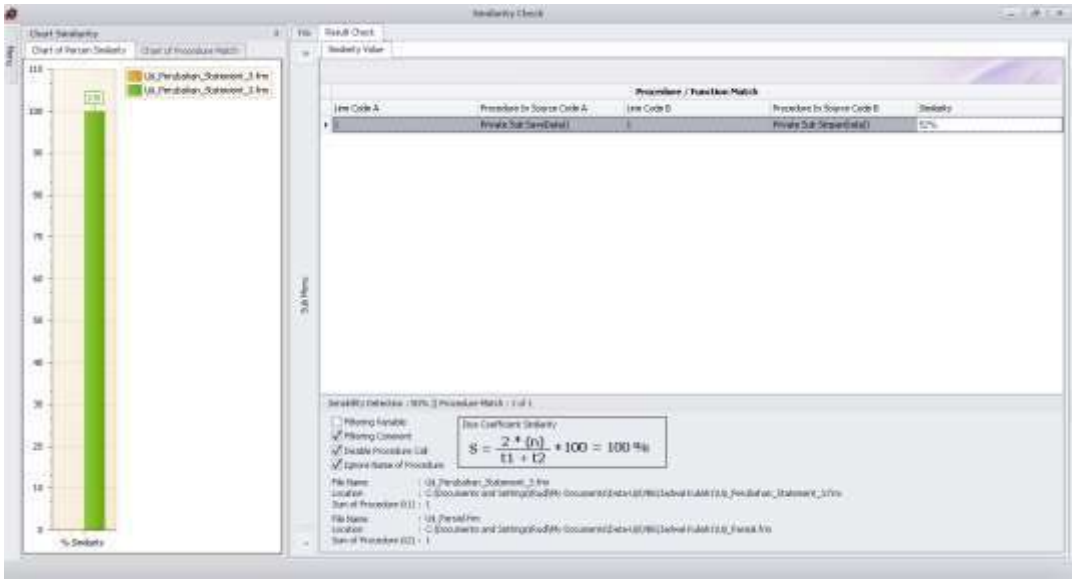
Gambar 6.16 Hasil pengujian kedua

3. Pada data yang sama dilakukan pengujian ketiga akan tetapi menggunakan parameter sensitifitas deteksi sebesar 50%. Gambar 6.17 merupakan parameter pengujian ketiga yang dilakukan.



Gambar 6.17 Parameter pengujian ketiga

Hasil pengujian ketiga pada Gambar 6.18, dengan parameter sensitifitas deteksi 50%, output sistem menemukan adanya similaritas sebesar 52%. Persentase similaritas yang dihasilkan sistem lebih besar dari nilai sensitifitas deteksi yang diberikan saat pengujian, sehingga pasangan procedure ini dianggap memiliki kesamaan. Similaritas kedua pasangan modul *source code* dihitung menggunakan metode dice coefficient menghasilkan similaritas sebesar 100%.



Gambar 6.18 Hasil pengujian ketiga

6.1.4 Pengujian modul *source code* pemrograman Visual Basic 6.0

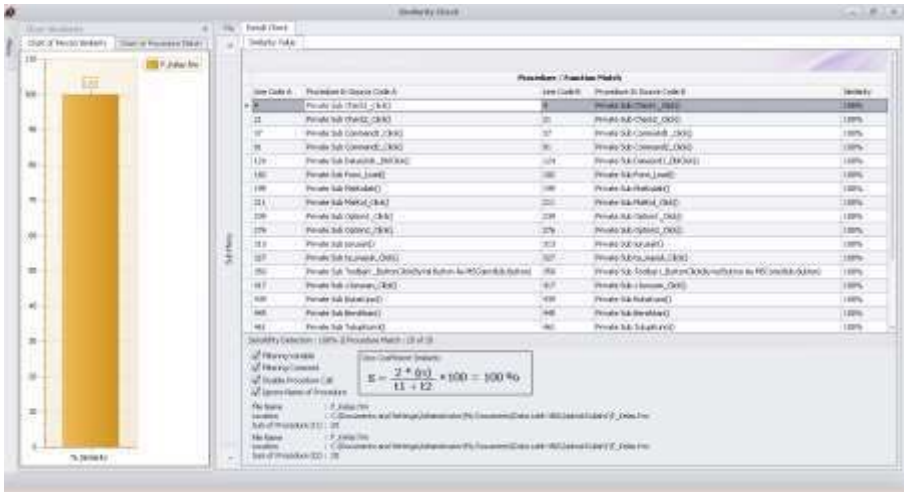
Kode uji VB-001

Pada kode uji VB-001, modul uji merupakan hasil *copy-paste* dari modul masternya, pengujian dilakukan dengan filtering variable dan parameter sensitifitas deteksi 100%, data pengujian dapat dilihat pada Tabel 6.1.

Tabel 6.1 Pengujian kode uji VB-001

Kode Uji	:	VB-001
Nama Modul Master	:	F_Kelas.frm
Nama Modul Uji	:	F_Kelas.frm
Keterangan tentang modul uji	:	Modul uji merupakan hasil copy-paste dari modul master
Parameter Proses	:	[×] Filtering variable [×] Filtering comment [×] Ignore name of procedure [×] Ignore procedure call Sensitifitas deteksi 100%
Jumlah Procedure Modul Master	:	20
Jumlah Procedure Modul Uji	:	20
Procedure Match	:	20
Output Sistem	:	Persentase kemiripan yang dihasilkan 100%
Kesimpulan Peneliti	:	Sistem mampu mengenali modul hasil copy-paste

Dari hasil pengujian yang dilakukan sistem pada Gambar 6.19, sistem mampu mengenali dua modul source code hasil *copy-paste* dengan output yang dihasilkan memiliki kemiripan 100%, hal ini berkesesuaian dengan data yang diuji, sehingga dapat disimpulkan output sistem dapat diterima.



Gambar 6.19 Output hasil kode uji VB-001

Kode uji VB-002

Pada kode uji VB-002 akan dilakukan beberapa modifikasi termasuk merubah urutan dari procedure, Gambar 6.20 menyatakan suatu procedure form_load pada modul master sebelum dilakukan modifikasi.

```
Private Sub Form_Load()
   Call Koneksi
   Frame1.Enabled = False
   Toolbar1.Buttons(2).Enabled = False
   Toolbar1.Buttons(3).Enabled = False
   Toolbar1.Buttons(4).Enabled = False
   Toolbar1.Buttons(5).Enabled = False
   Call TampilkanDataGrid
End Sub
```

Gambar 6.20 Code Snippets procedure form load

Gambar 6.21 adalah hasil modifikasi dari procedure form load pada modul master, modifikasi yang dilakukan ditunjukkan pada tulisan tebal, dimana telah dilakukan modifikasi berupa penambahan suatu komentar dalam procedure dan perubahan nama procedure yang dipanggil didalam body dari procedure tersebut seperti yang terlihat pada Gambar 6.21.

```
Private Sub Form_Load()
   'memanggil koneksi database
   Call Connect
   Frame1.Enabled = False
   Toolbar1.Buttons(2).Enabled = False
   Toolbar1.Buttons(3).Enabled = False
   Toolbar1.Buttons(4).Enabled = False
   Toolbar1.Buttons(5).Enabled = False
   Call ShowDataGrid
End Sub
```

Gambar 6.21 Code Snippets modifikasi procedure form load

Modifikasi nama procedure tanpa merubah isi dari procedure juga merupakan hal yang mudah dan umum dilakukan seperti yang terlihat pada Gambar 6.22 sebelum dilakukan modifikasi.

```
Private Sub TampilkanDataGrid()
Set rsdata = New ADODB.Recordset
strsql = "Select * from jurusan"
rsdata.Open strsql, conn, adOpenDynamic, adLockOptimistic

Set DataGrid1.DataSource = rsdata
Call SetGrid
End Sub
```

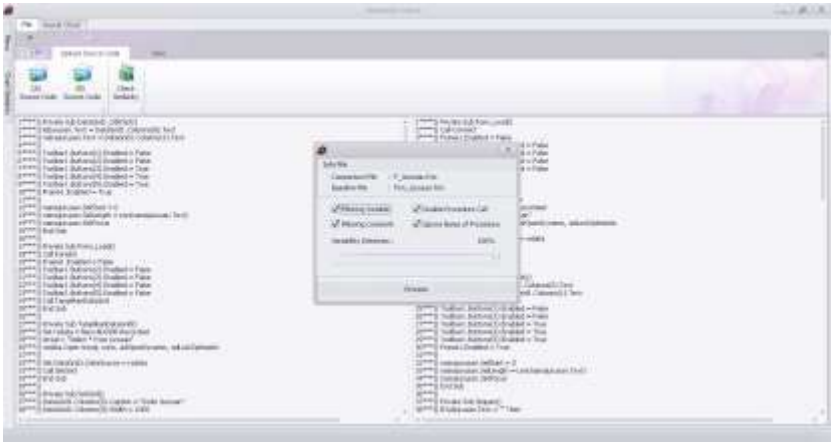
Gambar 6.22 Code Snippets procedure TampilkanDataGrid

Gambar 6.23 adalah contoh modifikasi dengan cara merubah nama procedure tanpa merubah isi dari procedure tersebut.

```
Private Sub ShowDataGrid()
  Set rsdata = New ADODB.Recordset
  strsql = "Select * from jurusan"
  rsdata.Open strsql, conn, adOpenDynamic, adLockOptimistic
  Set DataGrid1.DataSource = rsdata
  Call SetGridview
End Sub
```

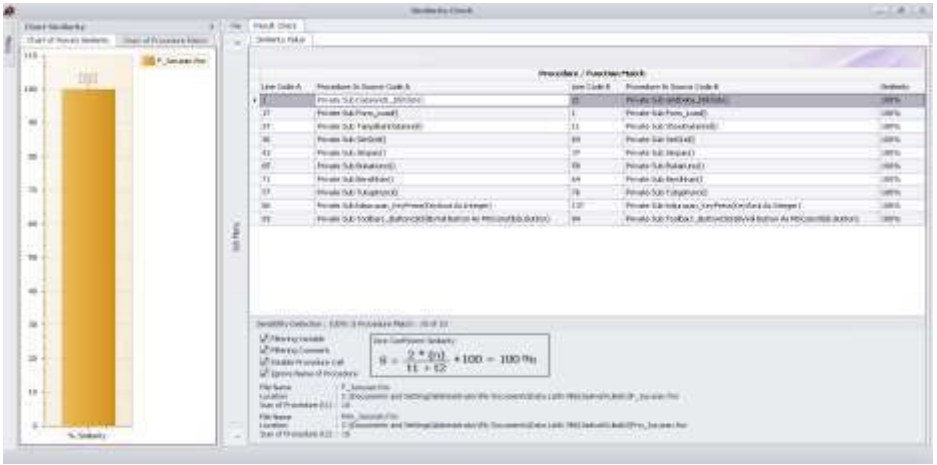
Gambar 6.23 Code Snippets procedure ShowDataGrid

Dari hasil pengujian yang dilakukan pada modul uji VB-002 dengan parameter pengujian seperti yang terlihat pada Gambar 6.24.



Gambar 6.24 Parameter pengujian VB-002

Dari hasil pengamatan output hasil pengujian pada Gambar 6.25, output yang dihasilkan oleh sistem mampu mengenali perubahan yang telah dilakukan, sehingga dapat disimpulkan hasil pengujian dapat diterima.



Gambar 6.25 Output hasil kode uji VB-002

Kode uji VB-003

Pada kode uji VB-003 akan dilakukan beberapa modifikasi termasuk merubah urutan dari procedure yang terdapat pada modul *source code*, Gambar 6.26 menyatakan suatu procedure *DataGrid1_DblClick* pada modul master sebelum dilakukan modifikasi.

```
Private Sub DataGrid1_DblClick()
Dim xNmJur As String
Dim xJenisKelamin As String
Dim xTahun1 As String
Dim xTahunKe1 As String
Dim xTahun2 As String
Dim xTahunKe2 As String

Npm.Text = DataGrid1.Columns(0).Text
Nama.Text = DataGrid1.Columns(1).Text
xTahun1 = Split(DataGrid1.Columns(2).Text, "-")(0)
xTahunKe1 = "01/01/" & Trim(xTahun1) & ""
DTPicker1.Value = xTahunKe1
xTahun2 = Split(DataGrid1.Columns(2).Text, "-")(1)
xTahunKe2 = "01/01/" & Trim(xTahun2) & ""
DTPicker2.Value = xTahunKe2
vNamaJurusan = DataGrid1.Columns(3).Text
Set rsdata = New ADODB.Recordset
str = "Select namajurusan from jurusan where kdjurusan = '" & vNamaJurusan & "'"
rsdata.Open str, conn, adOpenDynamic, adLockOptimistic
jurusan.Text = rsdata(0)
xJenisKelamin = DataGrid1.Columns(4).Text
If xJenisKelamin = "Pria" Then
   Option1.Value = True
ElseIf xJenisKelamin = "Wanita" Then
   Option2.Value = True
End If
tempatlahir.Text = DataGrid1.Columns(5).Text
tgllahir.Value = DataGrid1.Columns(6).Text
agama.Text = DataGrid1.Columns(7).Text
alamat.Text = DataGrid1.Columns(8).Text
   Toolbar1.Buttons(1).Enabled = False
   Toolbar1.Buttons(2).Enabled = False
   Toolbar1.Buttons(3).Enabled = True
   Toolbar1.Buttons(4).Enabled = True
   Toolbar1.Buttons(5).Enabled = True
   Frame1.Enabled = True
   Nama.SelStart = 0
   Nama.SelLength = Len(Nama.Text)
   Nama.SetFocus
End Sub
```

Gambar 6.26 Code Snippets procedure DataGrid1_DblClick

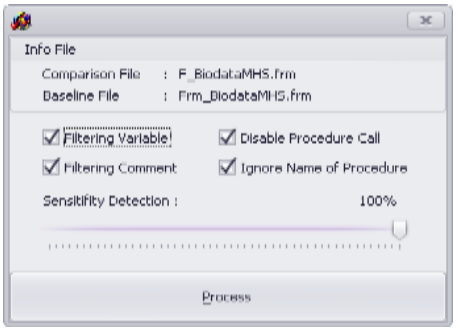
Gambar 6.27 adalah hasil modifikasi dari procedure *DataGrid1_DblClick* pada modul master, modifikasi yang dilakukan ditunjukkan pada tulisan tebal, dimana telah dilakukan modifikasi berupa perubahan nama variable beserta type data yang menyertainya, seperti yang terlihat pada Gambar 6.27.

```
Private Sub DataGrid1_DblClick()
Dim vNamaJurusan As String
Dim vJnsKel As String
Dim vThn1 As Date
Dim vThnKe1 As Date
Dim vThn2 As Date
Dim vThnKe2 As Date

Npm.Text = DataGrid1.Columns(0).Text
Nama.Text = DataGrid1.Columns(1).Text
vThn1 = Split(DataGrid1.Columns(2).Text, "-")(0)
vThnKe1 = "01/01/" & Trim(vThn1) & ""
DTPicker1.Value = vThnKe1
vThn2 = Split(DataGrid1.Columns(2).Text, "-")(1)
vThnKe2 = "01/01/" & Trim(vThn2) & ""
DTPicker2.Value = vThnKe2
vNamaJurusan = DataGrid1.Columns(3).Text
Set rsdata = New ADODB.Recordset
str = "Select namajurusan from jurusan where kdjurusan = '" & vNamaJurusan & "'"
rsdata.Open str, conn, adOpenDynamic, adLockOptimistic
jurusan.Text = rsdata(0)
vJnsKel = DataGrid1.Columns(4).Text
If vJnsKel = "Pria" Then
   Option1.Value = True
ElseIf vJnsKel = "Wanita" Then
   Option2.Value = True
End If
tempatlahir.Text = DataGrid1.Columns(5).Text
tgllahir.Value = DataGrid1.Columns(6).Text
agama.Text = DataGrid1.Columns(7).Text
alamat.Text = DataGrid1.Columns(8).Text
   Toolbar1.Buttons(1).Enabled = False
   Toolbar1.Buttons(2).Enabled = False
   Toolbar1.Buttons(3).Enabled = True
   Toolbar1.Buttons(4).Enabled = True
   Toolbar1.Buttons(5).Enabled = True
   Frame1.Enabled = True
   Nama.SelStart = 0
   Nama.SelLength = Len(Nama.Text)
   Nama.SetFocus
End Sub
```

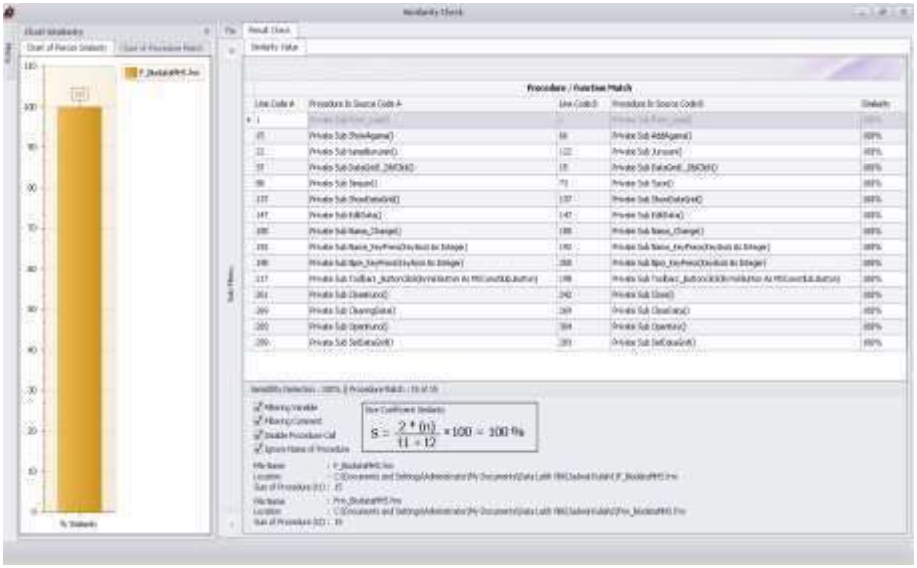
Gambar 6.27 Code Snippets modifikasi procedure DataGrid1_DblClick

Dari hasil pengujian yang dilakukan pada modul uji VB-003 dengan parameter pengujian seperti yang terlihat pada Gambar 6.28.



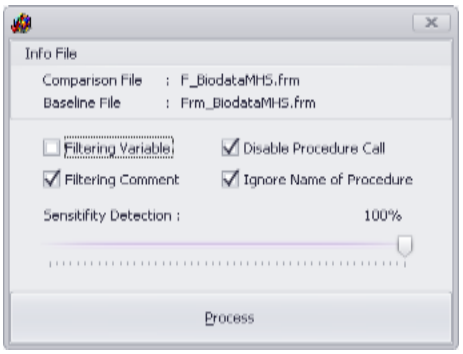
Gambar 6.28 Parameter pengujian pertama pada kode uji VB-003

Berdasarkan hasil pengamatan, output yang dihasilkan terhadap modifikasi yang telah dilakukan menunjukkan tingkat kemiripan 100%, ini berarti proses filtering variable yang diterapkan pada sistem berjalan dengan baik, dapat disimpulan bahwa sistem mampu mengenali perubahan yang telah dilakukan. Gambar 6.29 merupakan output sistem dari pengujian VB-003 menggunakan parameter filtering variable.



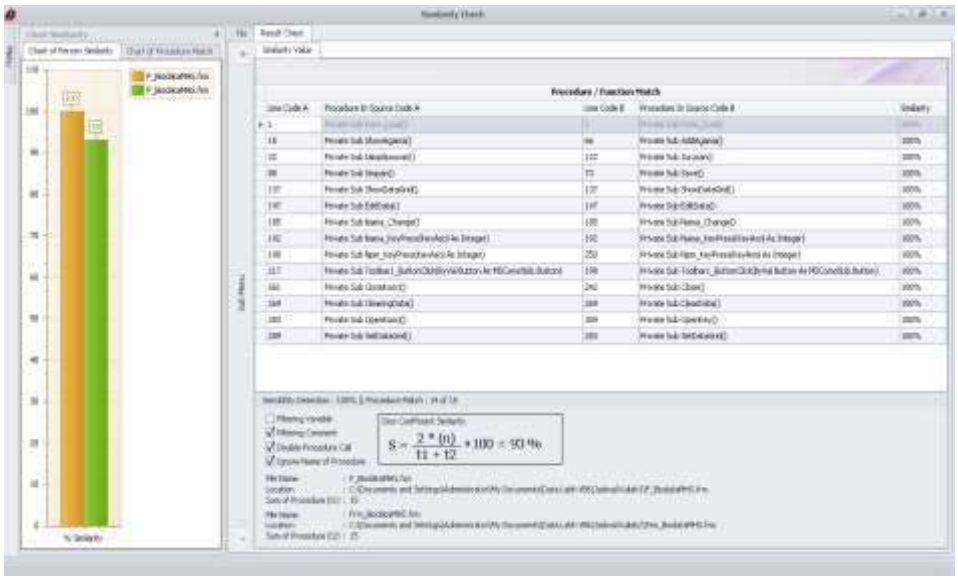
Gambar 6.29 Output pengujian pertama dari kode uji VB-003

Pengujian pada kode uji VB-003 dilakukan kembali dengan mengabaikan parameter filtering variable untuk melihat perbedaan yang semestinya dihasilkan dari output sistem, dengan parameter pengujian seperti yang terlihat pada Gambar 6.30 dengan mengabaikan parameter filtering variable.



Gambar 6.30 Parameter pengujian kedua pada kode uji VB-003

Output yang dihasilkan sistem pada pengujian kedua terdapat 14 procedure yang memiliki kesamaan dari total 15 procedure yang ada pada modul *source code*, persentase kemiripan yang dihasilkan sebesar 93%. 1 procedure dianggap tidak sama dikarenakan diabaikannya parameter filtering variable pada proses pengujian ini. Gambar 6.31 merupakan output sistem dari pengujian kode uji VB-003 dengan mengabaikan parameter filtering variable.



Gambar 6.31 Output pengujian kedua dari kode uji VB-003

6.1.5 Pengujian modul *source code* pemrograman VB.NET

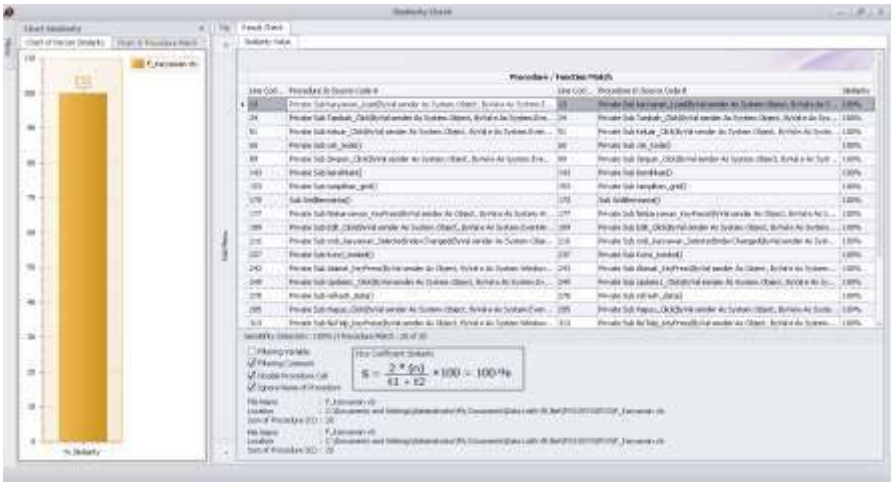
Kode uji VBNET-001

Pada kode uji VBNET-001, modul uji merupakan hasil *copy-paste* dari modul masternya, pengujian dilakukan dengan filtering variable dan parameter sensitifitas deteksi 100%, data pengujian dapat dilihat pada Tabel 6.2.

Tabel 6.2 Pengujian kode uji VBNET-001

Kode Uji	:	VBNET-001
Nama Modul Master	:	F_Karyawan.vb
Nama Modul Uji	:	F_Karyawan.vb
Keterangan tentang modul uji	:	Modul uji merupakan hasil copy-paste dari modul master
Parameter Proses	:	[ ] Filtering variable [×] Filtering comment [×] Ignore name of procedure [×] Ignore procedure call Sensitifitas deteksi 100%
Jumlah Procedure Modul Master	:	20
Jumlah Procedure Modul Uji	:	20
Procedure Match	:	20
Output Sistem	:	Persentase kemiripan yang dihasilkan 100%
Kesimpulan	:	Sistem mampu mengenali modul hasil copy-paste

Dari hasil pengujian yang dilakukan sistem pada ambar 6.32, sistem mampu mengenali dua modul source code hasil *copy-paste* dengan output yang dihasilkan memiliki kemiripan 100%, hal ini berkesesuaian dengan data yang diuji, sehingga dapat disimpulkan output sistem dapat diterima.



Gambar 6.32 Output sistem dari hasil kode uji VBNET-001

Kode uji VBNET-002

Pada kode uji VB-002 akan dilakukan beberapa modifikasi termasuk merubah urutan dari procedure, Gambar 6.33 menyatakan suatu procedure *karyawan_load* pada modul master sebelum dilakukan modifikasi.

```
Private Sub karyawan_Load(ByVal sender As Sistem.Object, ByVal e As Sistem.EventArgs)
Handles MyBase.Load
   Try
      EntryData.Enabled = False
      Call konek()
      Call tampilkan_grid()
      Call Kunci_tombol()
   Catch ex As Exception
      MsgBox(ex.Message)
   End Try
End Sub
```

Gambar 6.33 Code Snippets procedure karyawan_load

Gambar 6.34 adalah hasil modifikasi dari procedure *karyawan_load* pada modul master, modifikasi yang dilakukan ditunjukkan pada tulisan tebal, dimana telah dilakukan modifikasi berupa penambahan suatu komentar dalam procedure dan perubahan nama procedure yang dipanggil didalam body dari procedure tersebut seperti yang terlihat pada Gambar 6.34.

```
Private Sub karyawan_Load(ByVal sender As Sistem.Object, ByVal e As Sistem.EventArgs)
Handles MyBase.Load
   Try
      EntryData.Enabled = False
      'Buka Koneksi Database
      Call koneksi()
      'Tampilkan DataGrid
      Call showdatagrid()
      'Disable Tombol
      Call DisableButton()
   Catch ex As Exception
      MsgBox(ex.Message)
   End Try
End Sub
```

Gambar 6.34 Code Snippets modifikasi procedure karyawan_load

Modifikasi nama procedure tanpa merubah isi dari procedure juga merupakan hal yang mudah dan umum dilakukan seperti yang terlihat pada Gambar 6.35 sebelum dilakukan modifikasi.

```
Private Sub Keluar_Click(ByVal sender As Sistem.Object, ByVal e As Sistem.EventArgs)
Handles Keluar.Click
   Try
      Me.Dispose()
   Catch ex As Exception
      MsgBox(ex.Message)
   End Try
End Sub
```

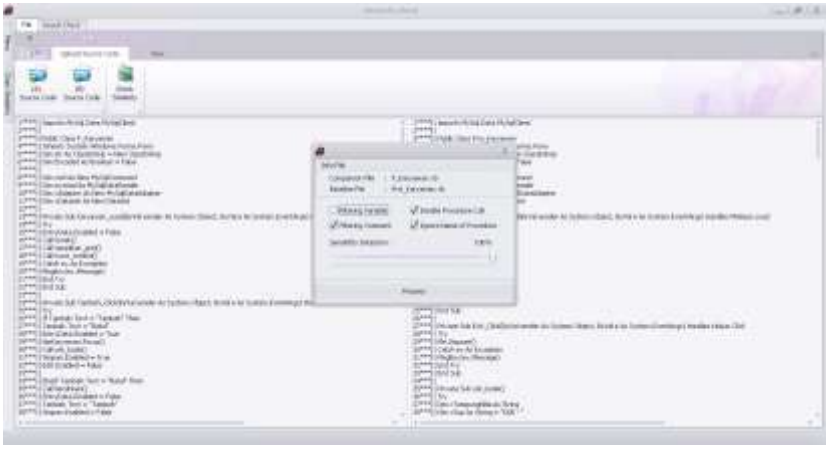
Gambar 6.35 Code Snippets procedure Keluar_Click

Gambar 6.36 adalah contoh modifikasi dengan cara merubah nama procedure tanpa merubah isi dari procedure tersebut.

```
Private Sub Exit_Click(ByVal sender As Sistem.Object, ByVal e As Sistem.EventArgs)
Handles Keluar.Click
   Try
      Me.Dispose()
   Catch ex As Exception
      MsgBox(ex.Message)
   End Try
End Sub
```

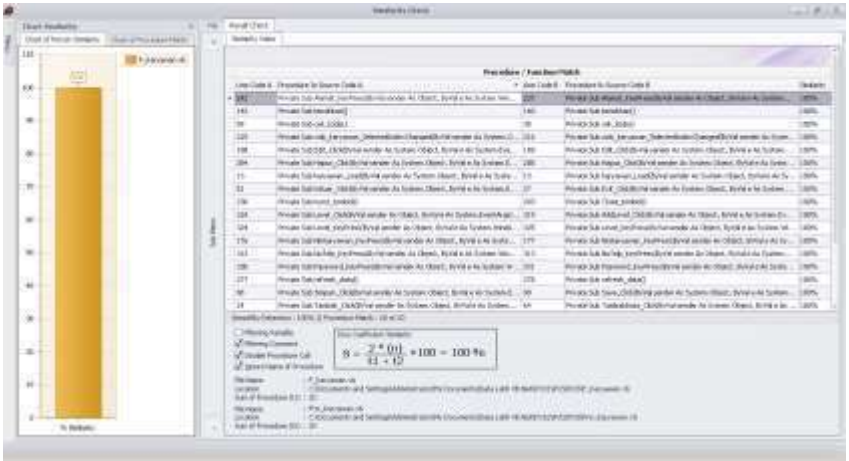
Gambar 6.36 Code Snippets procedure Exit_Click

Dari hasil pengujian yang dilakukan pada modul uji VBNET-002 dengan parameter pengujian seperti yang terlihat pada Gambar 6.37.



Gambar 6.37 Parameter pengujian VBNET-002

Hasil pengamatan output pengujian pada Gambar 6.38, output yang dihasilkan oleh sistem mampu mengenali perubahan yang telah dilakukan, sehingga dapat disimpulkan hasil pengujian dapat diterima.



Gambar 6.38 Output hasil kode uji VBNET-002

Kode uji VBNET-003

Pada kode uji VBNET-003 akan dilakukan beberapa modifikasi termasuk merubah urutan dari procedure yang terdapat pada modul source code, Gambar 6.39 menyatakan suatu procedure *cek_kode* pada modul master sebelum dilakukan modifikasi.

```
Private Sub cek_kode()
    Try
        Dim xTampungNilai As String
        Dim xSup As String = "KAR - "
        Dim xUrutan As String = "0000"
        Dim StrSQL As String = "Select no_anggota from karyawan"
        Dim cmd As New MySqlCommand(StrSQL, conn)
        Dim myread As MySqlDataReader
        myread = cmd.ExecuteReader
        If myread.HasRows = True Then
            For Each DataRow In myread
                xUrutan = DataRow(Trim("no_anggota"))
                xUrutan = Mid(xUrutan, 7, 1)
                xUrutan = Val(xUrutan) + 1
            Next
            xTampungNilai = Trim(xSup) & " " & xUrutan
        Else
            xTampungNilai = "KAR - 1"
        End If

        myread.Close()
        KdKaryawan.Text = Trim(xTampungNilai)

    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub
```

Gambar 6.39 Code Snippets procedure cek_kode

Gambar 6.40 adalah hasil modifikasi dari procedure *DataGrid1_DblClick* pada modul master, modifikasi yang dilakukan ditunjukkan pada tulisan tebal dan miring, dimana telah dilakukan modifikasi berupa perubahan nama variable beserta type data yang menyertainya, seperti yang terlihat pada Gambar 6.40.

```
Private Sub periksa_kode()
    Try
       Dim vTempValue As String
       Dim vSuplier As String = "KAR - "
       Dim vNoUrut As String = "0000"
       Dim StrSQL As String = "Select no_anggota from karyawan"
       Dim cmd As New MySqlCommand(StrSQL, conn)
       Dim myread As MySqlDataReader
       myread = cmd.ExecuteReader
       If myread.HasRows = True Then
          For Each DataRow In myread
              vNoUrut = DataRow(Trim("no_anggota"))
              vNoUrut = Mid(vNoUrut, 7, 1)
              vNoUrut = Val(vNoUrut) + 1
          Next
          vTempValue = Trim(vSuplier) & " " & vNoUrut
       Else
          vTempValue = "KAR - 1"
       End If
```

```
        myread.Close()
        KdKaryawan.Text = Trim(vTempValue)
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub
```

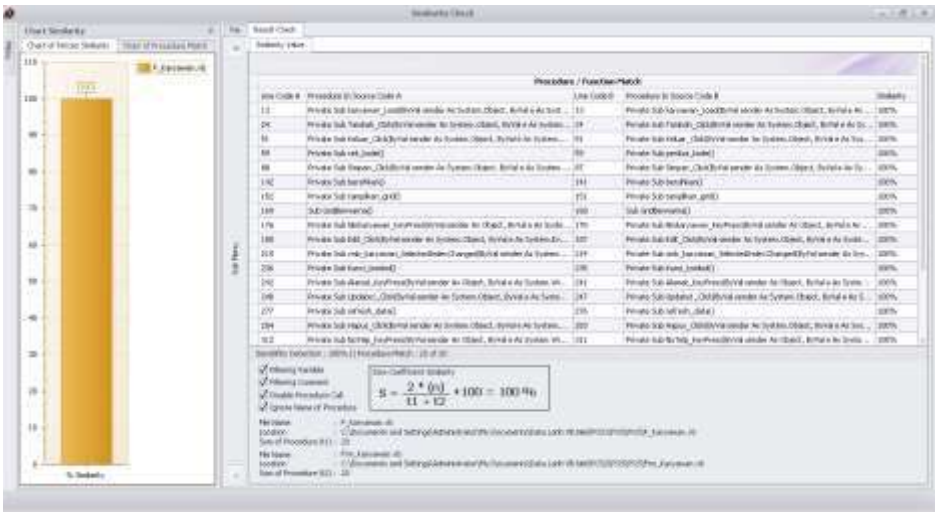
Gambar 6.40 Code Snippets modifikasi procedure periksa_code

Dari hasil pengujian yang dilakukan pada modul uji VBNET-003 dengan parameter pengujian seperti yang terlihat pada Gambar 6.41.

Gambar 6.41 Parameter pengujian pertama pada kode uji VBNET-003

Berdasarkan hasil pengujian dengan menggunakan semua parameter yang disediakan, output yang dihasilkan terhadap modifikasi yang telah dilakukan menunjukkan tingkat kemiripan 100%. Hal ini menunjukkan proses filtering variable yang diterapkan pada sistem berjalan dengan baik sehingga dapat disimpulkan bahwa sistem mampu mengenali perubahan yang telah dilakukan.

Gambar 6.42 merupakan output sistem dari pengujian VBNET-003 menggunakan parameter filtering variable.



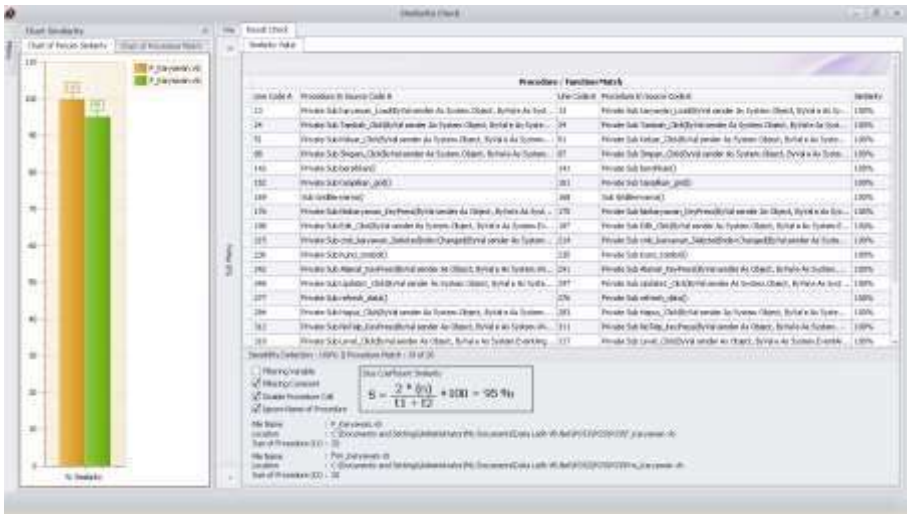
Gambar 6.42 Output pengujian pertama dari kode uji VBNET-003

Pengujian pada kode uji VBNET-003 dilakukan kembali dengan mengabaikan parameter filtering variable untuk melihat perbedaan yang semestinya dihasilkan dari output sistem, dengan parameter pengujian seperti yang terlihat pada Gambar 6.43 dengan mengabaikan parameter filtering variable.

Gambar 6.43 Parameter pengujian kedua pada kode uji VBNET-003

Output yang dihasilkan sistem dari pengujian kedua terdapat 19 procedure yang memiliki kesamaan dari total 20 procedure yang ada pada modul *source code*, persentase kemiripan yang dihasilkan sebesar 95%. Terdapat 1 procedure dianggap tidak sama dikarenakan diabaikannya parameter filtering variable pada proses pengecekan ini, sehingga dapat disimpulkan sistem mampu mendeteksi perubahan variable tetapi jika pada proses pengecekan mengabaikan parameter filtering variable maka sistem akan melakukan pengecekan tanpa mem-*filter* variable yang ada.

Gambar 6.44 merupakan output sistem dari pengujian kode uji VBNET-003 dengan mengabaikan parameter filtering variable.



Gambar 6.44 Output pengujian kedua dari kode uji VBNET-003

6.1.6 Pengujian modul *source code* pemrograman Borlan Delphi

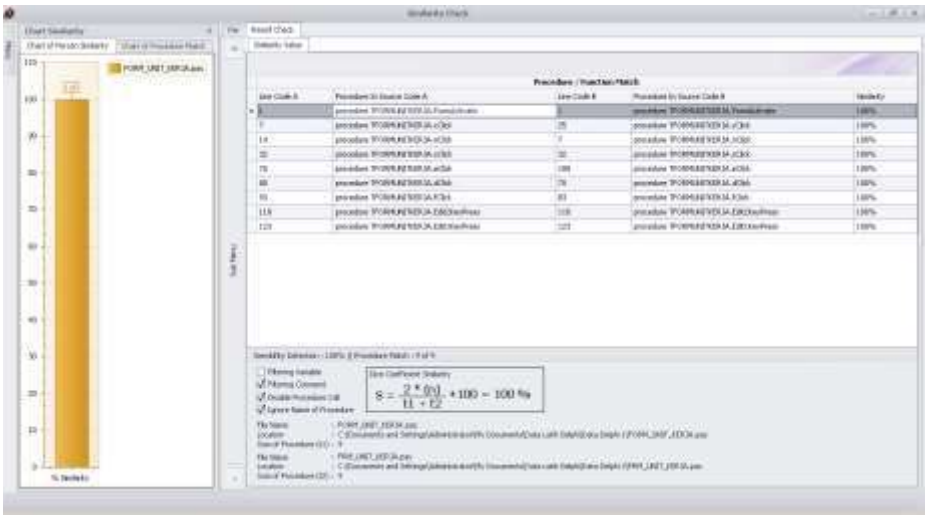
Kode uji DELPHI-001

Pada kode uji DELPHI-001, modul uji merupakan hasil *copy-paste* dari modul masternya, pengujian dilakukan tanpa parameter filtering variable dengan sensitifitas deteksi 100%, data pengujian dapat dilihat pada Tabel 6.3.

Tabel 6.3 Pengujian kode uji DELPHI-001

Kode Uji	:	DELPHI-001
Nama Modul Master	:	Form_Unit_Kerja.vb
Nama Modul Uji	:	Frm_Unit_Kerja.vb
Keterangan tentang modul uji	:	Modul uji merupakan hasil copy-paste dari modul master
Parameter Proses	:	[ ] Filtering variable [×] Filtering comment [×] Ignore name of procedure [×] Ignore procedure call Sensitifitas deteksi 100%
Jumlah Procedure Modul Master	:	9
Jumlah Procedure Modul Uji	:	9
Procedure Match	:	9
Output Sistem	:	Persentase kemiripan yang dihasilkan 100%
Kesimpulan	:	Sistem mampu mengenali modul hasil copy-paste

Hasil pengujian yang dilakukan sistem pada Gambar 6.45, sistem mampu mengenali dua modul source code hasil *copy-paste* dengan output yang dihasilkan memiliki kemiripan 100%, hal ini berkesesuaian dengan data yang diuji, sehingga dapat disimpulkan output sistem dapat diterima.



Gambar 6.45 Output sistem dari hasil kode uji DELPHI-001

Kode uji DELPHI-002

Pada kode uji DELPHI-002 akan dilakukan beberapa modifikasi termasuk merubah urutan dari procedure, Gambar 6.46 menyatakan suatu procedure *TFORMGolongan.nClick* pada modul master sebelum dilakukan modifikasi.

```
procedure TFORMGolongan.nClick(Sender: TObject);
begin
edit1.Text:='';
edit2.Text:='';
edit2.enabled:=false;
edit1.enabled:=true;
edit1.SetFocus;
s.Enabled :=true;
e.Enabled :=false;
d.Enabled:=false;
with DataModul1.ADOQuery1 do begin
SQL.Clear;
SQL.Add('Select * From Golongan');
Open;
DBGrid1.DataSource:=DataModul1.DataSource1;
end;
end;
```

Gambar 6.46 Code Snippets procedure TFORMGolongan.nClick

Gambar 6.47 merupakan hasil modifikasi dari procedure *TFORMGolongan.nClick* pada modul master, modifikasi yang dilakukan ditunjukkan pada tulisan tebal, dimana telah dilakukan modifikasi berupa penambahan komentar didalam procedure seperti yang terlihat pada Gambar 6.47.

```
procedure TFORMGolongan.nClick(Sender: TObject);
begin
//bersihkan textbox dan atur kondisi textbox
edit1.Text:='';
edit2.Text:='';
edit2.enabled:=false;
edit1.enabled:=true;
edit1.SetFocus;
s.Enabled :=true;
e.Enabled :=false;
d.Enabled:=false;
with DataModul1.ADOQuery1 do begin
SQL.Clear;
SQL.Add('Select * From Golongan');
Open;
DBGrid1.DataSource:=DataModul1.DataSource1;
end;
end;
```

Gambar 6.47 Code Snippets modifikasi procedure TFORMGolongan.nClick

Modifikasi nama procedure tanpa merubah isi dari procedure juga merupakan hal yang mudah dan umum dilakukan seperti yang terlihat pada Gambar 6.48 sebelum dilakukan modifikasi.

```
procedure TFORMGolongan.eClick(Sender: TObject);
begin
s.Enabled:=true;
e.Enabled:=false;
d.Enabled:=false;
edit2.Enabled:=true;
edit2.SetFocus;
end;
```

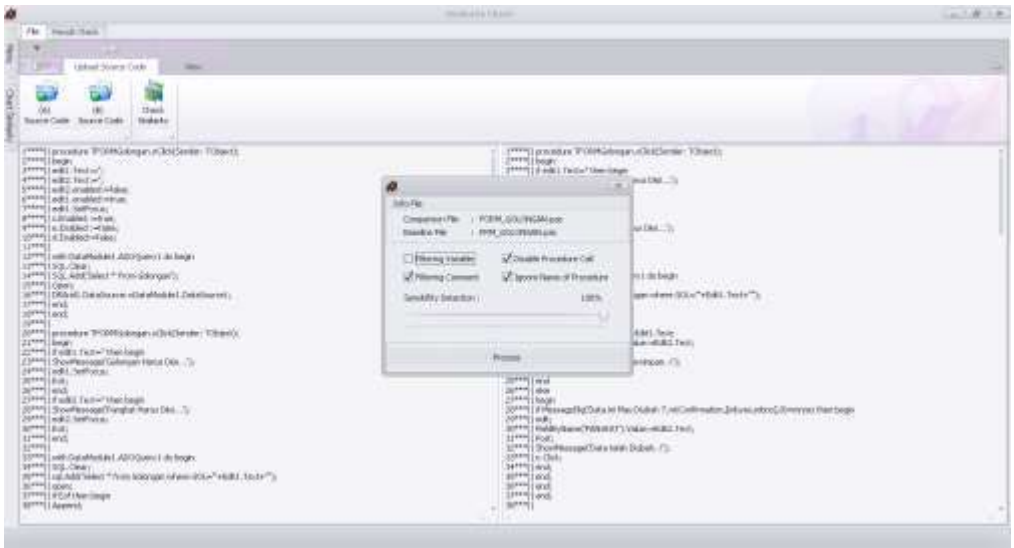
Gambar 6.48 Code Snippets procedure TFORMGolongan.eClick

Gambar 6.49 adalah contoh modifikasi dengan cara merubah nama procedure tanpa merubah isi dari procedure tersebut.

```
procedure TFORMGolongan.AturTombolClick(Sender: TObject);
begin
s.Enabled:=true;
e.Enabled:=false;
d.Enabled:=false;
edit2.Enabled:=true;
edit2.SetFocus;
end;
```

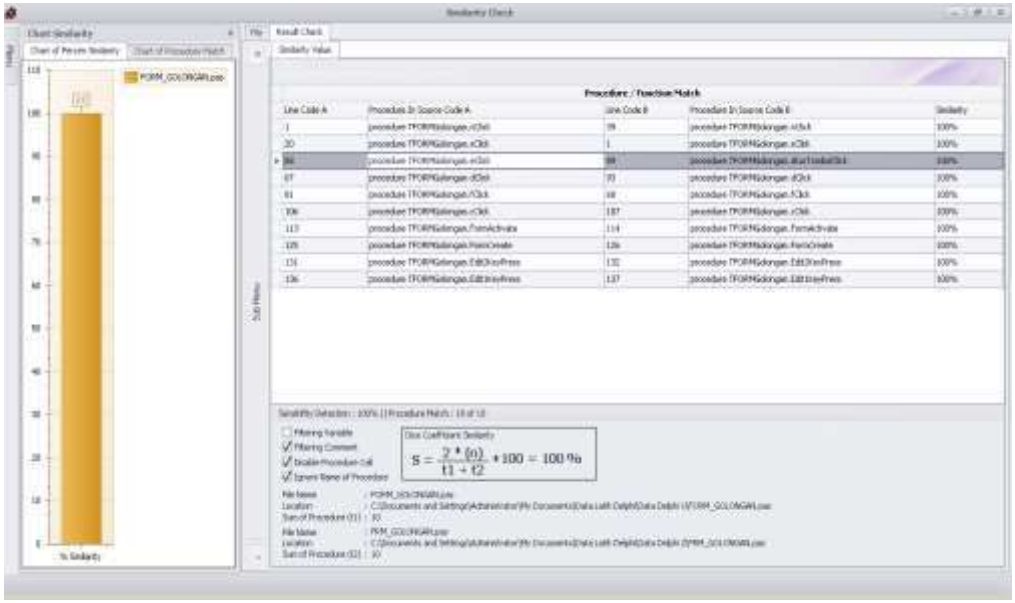
Gambar 6.49 Code Snippets procedure TFORMGolongan.AturTombolClick

Dari hasil pengujian yang dilakukan pada modul uji DELPHI-002 dengan parameter pengujian seperti yang terlihat pada Gambar 6.50.



Gambar 6.50 Parameter pengujian DELPHI-002

Dari hasil pengamatan output pengujian pada Gambar 6.51, output yang dihasilkan oleh sistem mampu mengenali perubahan yang telah dilakukan, sehingga dapat disimpulkan hasil pengujian dapat diterima.



Gambar 6.51 Output hasil kode uji DELPHI-002

Kode uji DELPHI-003

Pada kode uji DELPHI-003 akan dilakukan beberapa modifikasi termasuk merubah urutan dari procedure yang terdapat pada modul *source code*, Gambar 6.52 menyatakan suatu procedure *TFORMJABATAN.fClick* pada modul master sebelum dilakukan modifikasi.

```
procedure TFORMJABATAN.fClick(Sender: TObject);
var pesan:string;
begin
pesan:=inputbox('Cari Data','Masukkan Kode Jabatan atau Nama Jabatan yang akan dicari
:','');
with DataModul1.ADOQuery1 do begin
SQL.Clear;
sql.Add('Select * from JABATAN where kode_jabatan='''+pesan+'''or
Nama_Jabatan='''+pesan+'''');
Open;
if Eof then begin
ShowMessage('Data Tidak Ditemukan....!');
n.Click;
end
else
begin
Edit1.Text:=FieldByName('KODE_JABATAN').Value;
Edit2.Text:=FieldByName('nama_Jabatan').Value;
s.Enabled:=false;
e.Enabled:=true;
d.Enabled:=true;
edit2.Enabled:=false;
edit1.Enabled:=false;
end;
end;
end;
```

Gambar 6.52 Code Snippets procedure *TFORMJABATAN.fClick*

Gambar 6.53 adalah hasil modifikasi dari procedure *TFORMJABATAN.fClick* pada modul master, modifikasi yang dilakukan ditunjukkan pada tulisan tebal, dimana telah dilakukan modifikasi berupa perubahan nama variable, seperti yang terlihat pada Gambar 6.53.

```
procedure TFORMJABATAN.CariClick(Sender: TObject);
var kotak_masuk:string;
begin
kotak_masuk:=inputbox('Cari Data','Masukkan Kode Jabatan atau Nama Jabatan yang akan
dicari :','');
with DataModul1.ADOQuery1 do begin
SQL.Clear;
sql.Add('Select * from JABATAN where kode_jabatan='''+kotak_masuk+'''or
NAma_Jabatan='''+kotak_masuk+''''');
Open;
if Eof then begin
ShowMessage('Data Tidak Ditemukan....!');
n.Click;
end
else
begin
Edit1.Text:=FieldByName('KODE_JABATAN').Value;
Edit2.Text:=FieldByName('nama_Jabatan').Value;
s.Enabled:=false;
e.Enabled:=true;
d.Enabled:=true;
edit2.Enabled:=false;
edit1.Enabled:=false;
end;
end;
end;
```

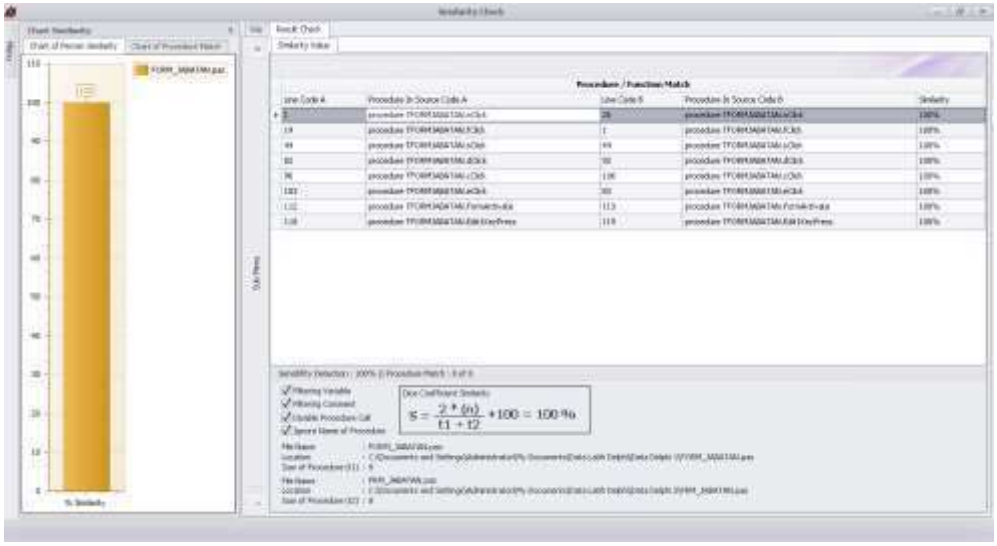
Gambar 6.53 Code Snippets modifikasi procedure *TFORMJABATAN.fClick*

Dari hasil pengujian yang dilakukan pada modul uji DELPHI-003 dengan parameter pengujian seperti yang terlihat pada Gambar 6.54.

Gambar 6.54 Parameter pengujian pertama pada kode uji DELPHI-003

Berdasarkan hasil pengujian dengan menggunakan semua parameter yang disediakan, output yang dihasilkan terhadap modifikasi yang telah dilakukan menunjukkan tingkat kemiripan 100%. Hal ini menunjukkan proses filtering variable yang diterapkan pada sistem berjalan dengan baik sehingga dapat disimpulkan bahwa sistem mampu mengenali perubahan yang telah dilakukan.

Gambar 6.55 merupakan output sistem dari pengujian DELPHI-003 menggunakan parameter filtering variable.



Gambar 6.55 Output pengujian pertama dari kode uji DELPHI-003

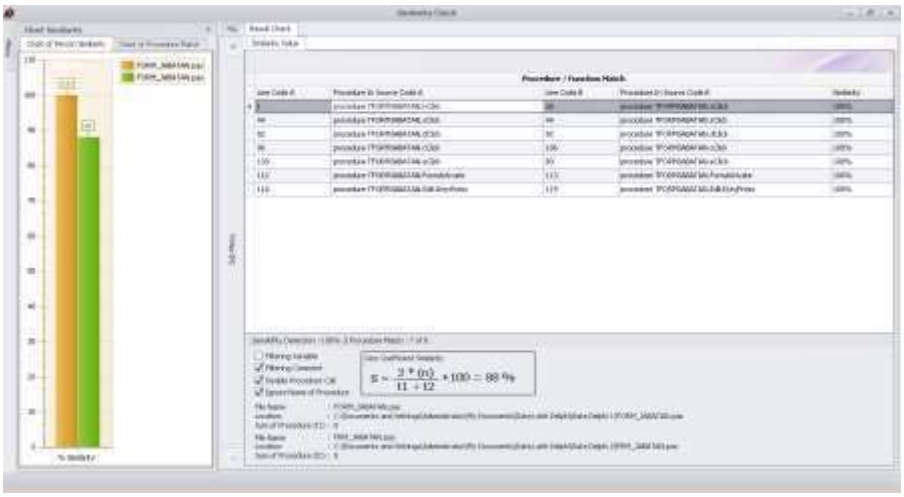
Pengujian pada kode uji DELPHI-003 dilakukan kembali dengan mengabaikan parameter filtering variable untuk melihat perbedaan yang semestinya dihasilkan oleh sistem, dengan parameter pengujian seperti yang terlihat pada Gambar 6.56 parameter pengujian dengan mengabaikan filtering variable.

Gambar 6.56 Parameter pengujian kedua pada kode uji DELPHI-003

Dari output yang dihasilkan sistem pada pengujian kedua terdapat 7 procedure yang memiliki kesamaan dari total 8 procedure yang ada pada modul *source code*, persentase kemiripan yang dihasilkan sebesar 88%. Terdapat 1 procedure dianggap tidak sama dikarenakan diabaikannya parameter filtering variable pada proses pengecekan ini, sehingga dapat disimpulkan sistem mampu mendeteksi perubahan variable tetapi jika pada proses pengecekan mengabaikan

parameter filtering variable maka sistem akan melakukan pengecekan tanpa mem-*filter* variable yang ada.

Gambar 6.57 merupakan output sistem dari pengujian kode uji DELPHI-003 dengan mengabaikan parameter filtering variable.



Gambar 6.57 Output pengujian kedua dari kode uji DELPHI-003

6.1.7 Pengujian modul *source code* pemrograman C#

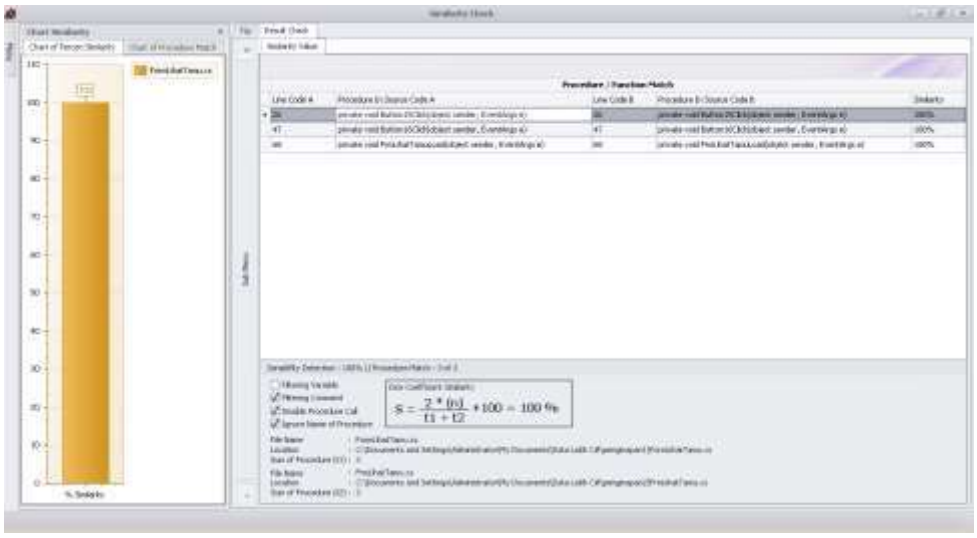
Kode uji CS-001

Pada kode uji CS-001, modul uji merupakan hasil *copy-paste* dari modul masternya, pengujian dilakukan tanpa parameter filtering variable dengan sensitifitas deteksi 100%, data pengujian dapat dilihat pada Tabel 6.4.

Tabel 6.4 Pengujian kode uji CS-001

Kode Uji	:	CS-001
Nama Modul Master	:	FormLihatTamu.cs
Nama Modul Uji	:	FrmLihatTamu.cs
Keterangan tentang modul uji	:	Modul uji merupakan hasil copy-paste dari modul master
Parameter Proses	:	[ ] Filtering variable [×] Filtering comment [×] Ignore name of procedure [×] Ignore procedure call Sensitifitas deteksi 100%
Jumlah Procedure Modul Master	:	3
Jumlah Procedure Modul Uji	:	3
Procedure Match	:	3
Output Sistem	:	Persentase kemiripan yang dihasilkan 100%
Kesimpulan	:	Sistem mampu mengenali modul hasil copy-paste

Hasil pengujian yang dilakukan sistem pada Gambar 6.58, sistem mampu mengenali dua modul source code hasil *copy-paste* dengan output yang dihasilkan memiliki kemiripan 100%, hal ini berkesesuaian dengan data yang diuji, sehingga dapat disimpulkan output sistem dapat diterima.



Gambar 6.58 Output sistem hasil kode uji CS-001

Kode uji CS-002

Pada kode uji CS-002 akan dilakukan beberapa modifikasi, Gambar 6.59 menyatakan suatu procedure *FormShowTypeLoad* pada modul master sebelum dilakukan modifikasi.

```
private void FormShowTypeLoad(object sender, EventArgs e)
{
     string koneksi = "server=localhost;database=db_penginapan;uid=root;password=";
                      MySqlConnection koneksiB = new MySqlConnection(koneksi);
                      MySqlCommand com = new MySqlCommand("call LihatTipe()",koneksiB);
                      MySqlDataAdapter adap = new MySqlDataAdapter(com);
     tampil.Columns.Clear();
     DataTable table = new DataTable();
     adap.Fill(table);
     tampil.DataSource = table;
     tampil.AllowUserToAddRows = false;
     tampil.ReadOnly = true;
     tampil.Columns[0].HeaderText = "ID Tipe Kamar";
     tampil.Columns[0].Width = 100;
     tampil.Columns[1].HeaderText = "Nama Tipe Kamar";
     tampil.Columns[1].Width = 170;
}
```

Gambar 6.59 Code Snippets procedure FormShowTypeLoad

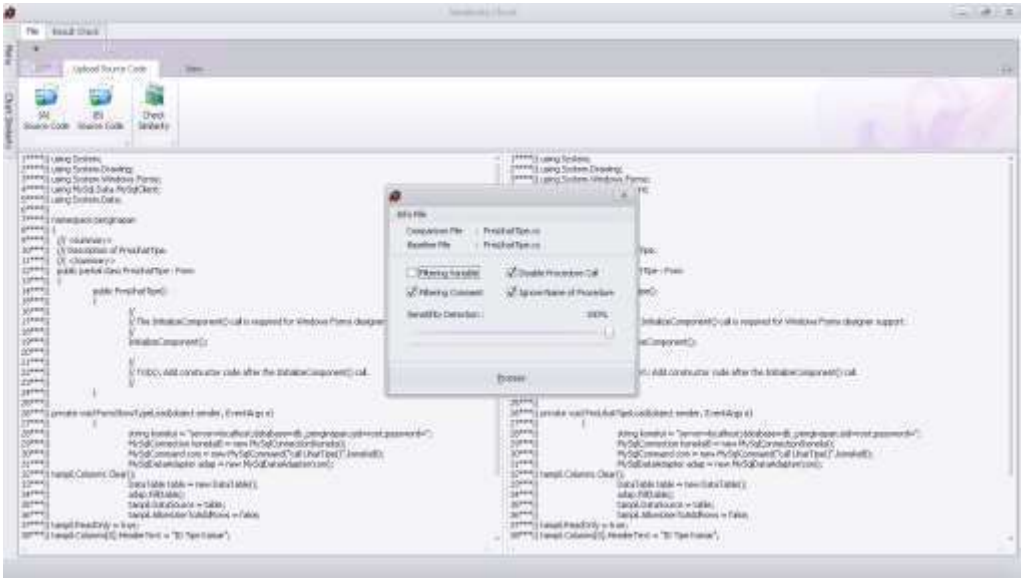
Gambar 6.60 merupakan hasil modifikasi dari procedure *FormShowTypeLoad* pada modul master, modifikasi yang dilakukan ditunjukkan pada tulisan tebal, dimana telah dilakukan modifikasi berupa perubahan nama procedure serta

penambahan komentar didalam procedure tanpa merubah isi dari procedure seperti yang terlihat pada Gambar 6.60.

```
private void FrmLihatTipeLoad(object sender, EventArgs e)
{
     // buka koneksi database
     string koneksi = "server=localhost;database=db_penginapan;uid=root;password=";
                      MySqlConnection koneksiB = new MySqlConnection(koneksi);
                      MySqlCommand com = new MySqlCommand("call LihatTipe()",koneksiB);
                      MySqlDataAdapter adap = new MySqlDataAdapter(com);
     tampil.Columns.Clear();
     DataTable table = new DataTable();
     adap.Fill(table);
     tampil.DataSource = table;
     tampil.AllowUserToAddRows = false;
     tampil.ReadOnly = true;
     tampil.Columns[0].HeaderText = "ID Tipe Kamar";
     tampil.Columns[0].Width = 100;
     tampil.Columns[1].HeaderText = "Nama Tipe Kamar";
     tampil.Columns[1].Width = 170;
}
```

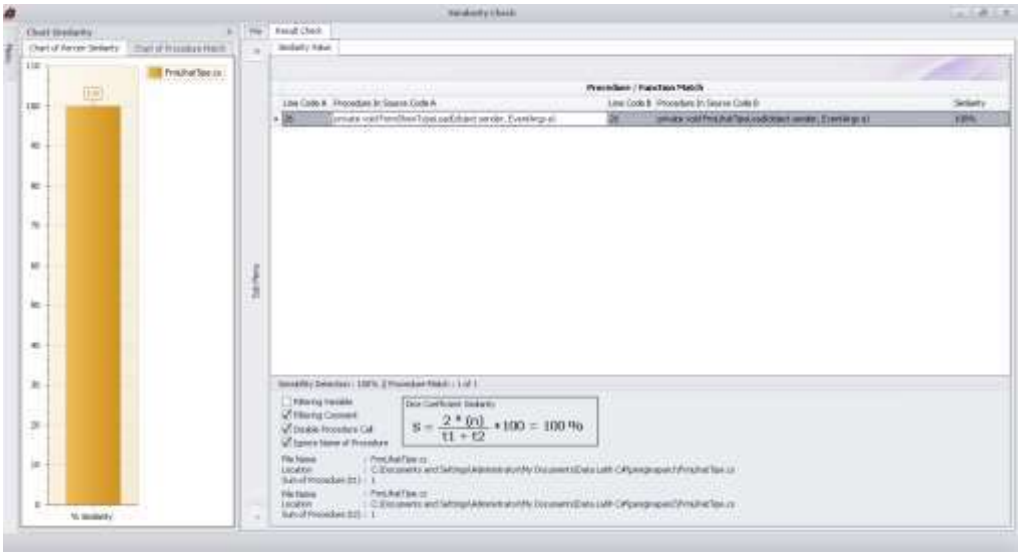
Gambar 6.60 Code Snippets modifikasi procedure FrmLihatTipeLoad

Dari hasil pengujian yang dilakukan pada modul uji CS-002 dengan parameter pengujian seperti yang terlihat pada Gambar 6.61.



Gambar 6.61 Parameter pengujian CS-002

Dari hasil pengamatan output pengujian pada Gambar 6.62, output yang dihasilkan oleh sistem mampu mengenali perubahan yang telah dilakukan, sehingga dapat disimpulkan hasil pengujian dapat diterima.



Gambar 6.62 Output hasil kode uji CS-002

Kode uji CS-003

Pada kode uji CS-003 akan dilakukan beberapa modifikasi termasuk merubah urutan dari procedure, Gambar 6.63 menyatakan suatu procedure *FrmTambahKamarLoad* pada modul master sebelum dilakukan modifikasi.

```
private void FrmTambahKamarLoad(object sender, EventArgs e)
{
       string koneksi = "server=localhost;database=db_penginapan;uid=root;password=";
                        MySqlConnection koneksiB = new MySqlConnection(koneksi);
                        MySqlCommand com = new MySqlCommand(TipeKamar(),koneksiB);
                        MySqlDataAdapter adap = new MySqlDataAdapter(com);
                        DataTable table = new DataTable();
                        adap.Fill(table);
        foreach(DataRow kolom in table.Rows)
        {
                string id_tipe = kolom["id_tipe_kamar"].ToString();
                string nama = kolom["nama"].ToString();
                string gabung = id_tipe +" - "+ nama;
                cmbCariTipeKamar.Items.Add(gabung);
        }
}
```

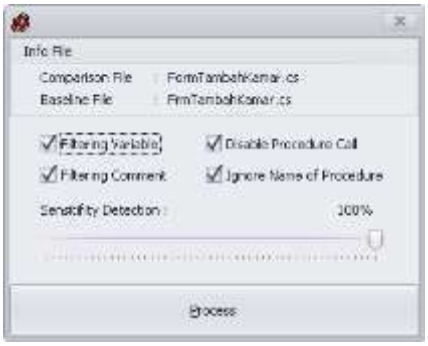
Gambar 6.63 Code Snippets procedure FrmTambahKamarLoad

Gambar 6.64 merupakan hasil modifikasi dari *FrmTambahKamarLoad* pada modul master, modifikasi yang dilakukan ditunjukkan pada tulisan tebal, dimana telah dilakukan modifikasi berupa perubahan nama variable seperti yang terlihat pada Gambar 6.64.

```
private void FrmTambahKamarLoad(object sender, EventArgs e)
{
     string connected = "server=localhost;database=db_penginapan;uid=root;password=";
                        MySqlConnection koneksiB = new MySqlConnection(connected);
                        MySqlCommand com = new MySqlCommand(TipeKamar(),koneksiB);
                        MySqlDataAdapter adap = new MySqlDataAdapter(com);
                        DataTable table = new DataTable();
                        adap.Fill(table);
     foreach(DataRow kolom in table.Rows)
     {
              string type_ID = kolom["id_tipe_kamar"].ToString();
              string name = kolom["name"].ToString();
              string datapencarian = type_ID +" - "+ name;
              cmbCariTipeKamar.Items.Add(datapencarian);
     }
}
```

Gambar 6.64 Code Snippets modifikasi procedure FrmTambahKamarLoad

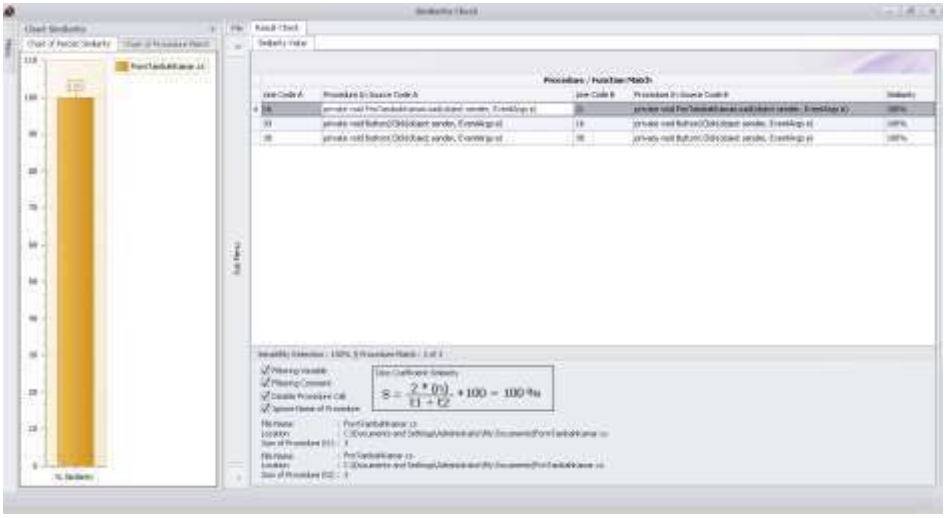
Dari hasil pengujian yang dilakukan pada modul uji CS-003 dengan parameter pengujian seperti yang terlihat pada Gambar 6.65.



Gambar 6.65 Parameter pengujian pertama pada kode uji CS-003

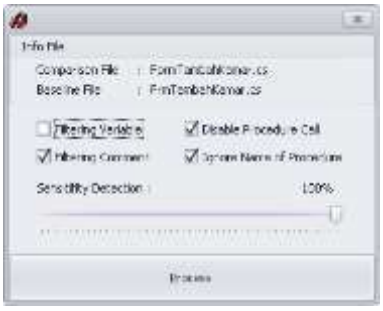
Berdasarkan hasil pengujian dengan menggunakan semua parameter yang disediakan, pada Gambar 6.66 output yang dihasilkan terhadap modifikasi yang telah dilakukan menunjukkan tingkat kemiripan 100%. Hal ini menunjukkan proses filtering variable yang diterapkan pada sistem berjalan dengan baik sehingga dapat disimpulkan bahwa sistem mampu mengenali perubahan yang telah dilakukan.

Gambar 6.66 merupakan output sistem dari pengujian CS-003 menggunakan parameter filtering variable.



Gambar 6.66 Output pengujian pertama dari kode uji CS-003

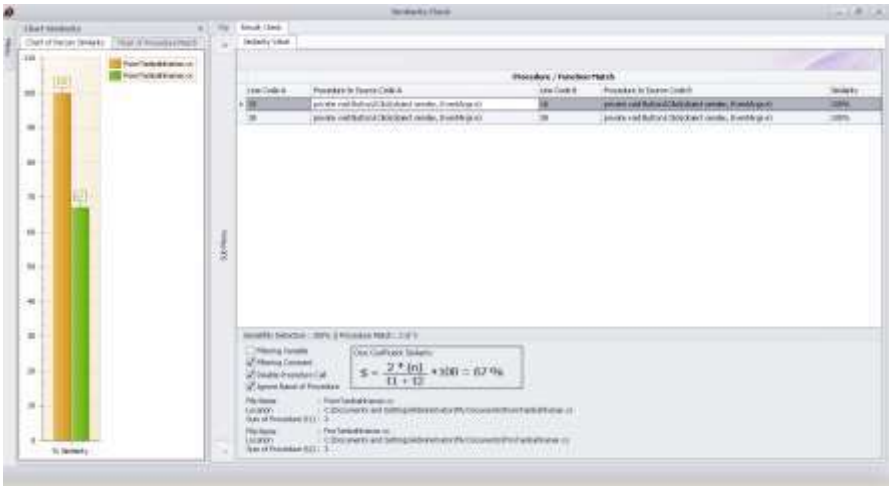
Pengujian pada kode uji CS-003 dilakukan kembali dengan mengabaikan parameter filtering variable untuk melihat perbedaan yang semestinya dihasilkan oleh sistem, dengan parameter pengujian seperti yang terlihat pada Gambar 6.67 dengan mengabaikan filtering variable.



Gambar 6.67 Parameter pengujian kedua pada kode uji CS-003

Output yang dihasilkan sistem pada pengujian kedua terdapat 2 procedure yang memiliki kesamaan dari total 3 procedure yang ada pada modul source code, persentase kemiripan yang dihasilkan sebesar 67%. Terdapat 1 procedure dianggap tidak sama dikarenakan diabaikannya parameter filtering variable pada proses pengecekan ini, sehingga dapat disimpulkan sistem mampu mendeteksi perubahan variable tetapi jika pada proses pengecekan mengabaikan parameter filtering variable maka sistem akan melakukan pengecekan tanpa mem-*filter* variable yang ada.

Gambar 6.68 merupakan output sistem dari pengujian kode uji CS-003 dengan mengabaikan parameter filtering variable.



Gambar 6.68 Output pengujian kedua dari kode uji CS-003

6.2 Catatan *Running Time* pada Proses Pengecekan

Catatan *running time* saat proses pengecekan dilakukan guna mengetahui waktu yang dibutuhkan oleh sistem dalam melakukan suatu proses pengecekan, *running time* yang ada pada Tabel 6.5 berkaitan dengan data hasil pengujian pada sub bab 6.1.4, 6.1.5, 6.1.6 dan 6.1.7.

Tabel 6.5 *Running Time* Proses Pengecekan

No	Kode Uji	Source Code A		Source Code B		Waktu Proses (detik)
		Jumlah Baris Kode	Jumlah Procedure	Jumlah Baris Kode	Jumlah Procedure	
1	VB-001	534	20	534	20	1.1
2	VB-002	144	10	142	10	0.71
3	VB-003	308	15	308	15	0.75
4	VBNET-001	336	20	336	20	0.84
5	VBNET-002	339	20	336	20	0.89
6	VBNET-003	336	20	336	20	0.86
7	DELPHI-001	146	9	146	9	0.77
8	DELPHI-002	163	10	163	10	0.82
9	DELPHI-003	142	8	143	8	0.81
10	CS-001	88	3	88	3	0.96
11	CS-002	44	1	44	1	0.55
12	CS-003	68	3	68	3	0.79

Gambar 6.69 merupakan grafik *running time* proses pengecekan berdasarkan data pengujian yang ada pada sub bab 6.1.4, 6.1.5, 6.1.6 dan 6.1.7.

Gambar 6.69 Grafik *running time* proses pengecekan

BAB VII
KESIMPULAN

7.1 Kesimpulan

Dari penelitian yang telah dilakukan terdapat beberapa kesimpulan yaitu :

1. Pembentukan token yang berbeda antara penelitian terdahulu dengan penelitian saat ini (lihat kembali sub bab 4.4) serta algoritma RKRGST yang diterapkan pada sistem ini, mampu mengenali perubahan statement maupun urutan statement serta mampu pula mengenali *source code* uji yang isinya sebagian berasal dari modul *source code* pembandingnya.
2. a. Pengujian sistem terhadap *source code* hasil modifikasi parsial (modifikasi yang dilakukan dengan cara mengambil sebagian isi *source code*) menghasilkan beberapa hal sebagai berikut :
   i. Jika panjang *syntax procedure* yang terdapat pada *source code* uji lebih pendek dari *syntax procedure* yang terdapat pada *source code* pembandingnya, sebaiknya menggunakan seting sensitivitas deteksi sebesar 100%. Jika dalam pengujian didapat hasil kemiripan sebesar 100%, maka dapat disimpulkan bahwa isi *syntax procedure* yang diuji secara keseluruhan memiliki kesamaan pada *syntax procedure* yang terdapat pada *source code* pembandingnya.
   ii. Jika *syntax procedure* yang akan diuji lebih panjang dari *source code* pembandingnya sebaiknya tidak menggunakan seting sensitivitas deteksi sebesar 100%. Karena maksimal persentase kemiripan antar procedure yang diuji tidak akan mencapai 100%. Sehingga ada baiknya untuk kasus seperti ini menggunakan nilai sensitivitas deteksi kurang dari 100%.

   b. Jika dalam melakukan pengujian menggunakan sensitivitas deteksi 100% dan pada pengujian didapat hasil similaritas sebesar 100%, maka terdapat beberapa kemungkinan mengenai *source code* yang diuji :
   i. Dengan tidak menggunakan filtering variable saat pengujian, *source code* tersebut merupakan hasil *copy-paste*

ii. Dengan menggunakan filtering variable saat pengujian, *source code* tersebut merupakan hasil *copy-paste* tetapi telah dilakukan modifikasi variable. Kepastian bahwa, apakah benar telah dilakukan perubahan variable, dapat dilakukan pengujian kembali pada *source code* ini dengan mengabaikan filtering variable. Jika pada pengujian dengan mengabaikan filtering variable hasil similaritas yang didapat berbeda seperti pada saat menggunakan filtering variable, maka dapat dipastikan telah terjadi modifikasi perubahan nama variable maupun type data yang terdapat pada *source code* yang diuji.

iii. *Source code* tersebut merupakan hasil *copy-paste* yang disertai perubahan urutan statement didalamnya.

3. Waktu proses yang dibutuhkan oleh sistem untuk menghasilkan output bergantung pada banyaknya baris kode program yang terdapat pada modul *source code*. Semakin banyak baris kode program yang ada maka akan semakin lama waktu proses yang dibutuhkan untuk menghasilkan output.

7.2 Saran

1. Guna pengembangan sistem yang memiliki proses pengecekan yang lebih komprehensif tanpa mengutamakan kecepatan hasil proses deteksi, dapat ditambahkan mekanisme deteksi berupa graf eksekusi, data flow dan lain sebagainya.

DAFTAR PUSTAKA

Aiken, A., (1998). MOSS (Measure Of Software Similarity) plagiarism detection sistem. http://www.cs.berkeley.edu/~moss, diakses 21 Januari 2012.

Bull, J., Collins, C., Coughlin, E. And Sharp, D., (2001). Technical review of plagiarism detection software report, CAA, University of Luton.

Clough, P., (2000). *Plagiarism in natural and programming languages: an overview of current tools and technologies*, Department of Computer Science, University of Sheffield.

Culwin, F., MacLeod, A. and Lancaster T., (2001). Source code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools, Technical Report No. SBU-CISM-01-02, South Bank University.

Faidhi, J. A., Robinson, S. K., (1987). *An empirical approach for detecting program Similarity within a university programming environment*, Computers and Education, Vol. 11, No. 1, Pp. 11-19.

Firdaus, H. B., (2003). Deteksi Plagiat Dokumen Menggunakan Algoritma Rabin Karp, Jurnal Ilmu Komputer dan Teknologi Informasi Vol III No.2.

Goel, S., Rao, D., (2003). *Plagiarism and its Detection in Programming Languages*, Technical Report, Department of Computer Science and Information Technology, Stanford University.

Karp, R. M., Rabin, M. O., (1987). *Efficient Randomized Pattern-Matching Algorithms*. IBM Journal of Research and Development 31(2), p249-260.

Kustanto, C., Liem, I., (2009). "*Automatic Source Code Plagiarism Detection*" 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing. pp.481-486.

Prechelt, L., Malpohl, G., and Phlippsen, M., (2000). *JPlag: Finding plagiarisms among a set of programs*, Technical Report 2000-1, Fakultat fur Informatik Universitat Karlsruhe.

Schleimer, S., Daniel, S. W., and Aiken, A., (2003). "*Winnowing: Local Algorithms for Document Fingerprinting*", SIGMOD 2003. San Diego, CA.

Wagner, N. R., (2000). *Plagiarism by Student Programmers*. The University of Texas at San Antonio, Division Computer Science, San Antonio, TX 78249, USA.

Whale, G., (1990). *Identification of Program Similarity in Large Populations*. The Computer Journal, Vol. 33, No 2, Pages: 140-146.

Wise, M. J., (1993). String Similarity via Greedy String Tiling and Running Karp-Rabin Matching. Technical Report, Department of Computer Science, University of Sydney, Australia.

Wise, M. J., (1996). "*YAP3: Improved Detection of Similarities in Computer Program and Other Texts*," Proceedings of SIGCSE, Volume: 28, Issue: 1, Publisher: ACM New York, NY, USA, Pages: 130-134.

Verco, K. L., Wise, M. J., (2006). *Comparing Structure and Attribute-Counting System*. Technical Report, Department of Computer Science, University of Sydney, Australia.

LAMPIRAN

Cara kerja algoritma Running Karp-Rabin Greedy String Tiling

Berikut akan dijelaskan mengenai cara kerja dari algoritma RKR-GST dalam proses pencarian string, sebagai contoh misalkan terdapat dua string teks yang akan dibandingkan yaitu string teks T dan P.

String Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

String Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Pada contoh ini, input parameter *search-length (s)* di *set* = 2, dengan begitu, pasangan *substring* sama yang mempunyai panjang lebih kecil dari 2 akan diabaikan. Jika 1 dipilih sebagai *search-length (s),* setiap pasangan token yang sama akan dianggap sebagai hasil plagiarisme. Hal itu tentunya berakibat pada hasil deteksi yang kurang akurat karena pada umumnya satu *statement* yang serupa sering ditemui pada *source code*. Kasus tersebut bukanlah plagiarisme melainkan kesamaan yang tidak disengaja. Jika *search-length (s)* yang dipilih bernilai besar maka akurasi hasil deteksi juga kurang optimal dikarenakan akan ada banyak pasangan *substring* sama yang terabaikan.

Berikut ini dijelaskan kasus-kasus yang terjadi ketika eksekusi pembandingan *token string* P dan T.

5. Inisialisasi nilai awal s=min_max_match, Pada Top Level RKRGST dengan nilai s=2, stop=false, kemudian panggil *Scanpattern*
2. Pada Fase *scanpattern* untuk t=1 sepanjang s sampai dengan panjang dari teks - s, maka dilakukan pengumpulan substring untuk membuat nilai hash dari teks sepanjang s. pada algoritma di fase *scanpattern* perlu diperhatikan apakah sebuah string telah ditandai, karena jika sebuah string telah ditandai maka string tersebut telah dimiliki oleh suatu *tile* sehingga tidak dapat

dibentuk substring dari sebuah string yang telah ditandai, pada contoh ini untuk pembentukan substring di fase pertama *scanpattern* hal ini tidak sulit dilakukan karena tidak ada mark pada string sepanjang teks, maka setiap 2 karakter dapat langsung dibentuk substring dan dihitung nilai hash-nya dan kemudian disimpan kedalam *hashtable*. Contoh : BA dihash, lalu simpan kedalam *hashtable*, AN dihash , lalu simpan kedalam *hashtable*, NA dihash lalu simpan kedalam *hashtable*, sampai dengan akhir teks.

3. Untuk mempermudah contoh, kita umpamakan terlebih dahulu bahwa nilai *hash* sama dengan nilai substring nya. Setelah substring pada teks telah terbentuk dan nilai hash telah disimpan kedalam hashtable, selanjutnya pada string pola dilakukan hal yang sama seperti pada string teks hanya saja pada string pola setiap satu substring yang terbentuk dan telah dihitung nilai hash nya akan langsung dicocokkan nilai hash nya dengan nilai hash yang ada di *hashtable* dari string T, Sebagai contoh Untuk P=1 yaitu substring PB di-hash lalu dicocokkan dengan semua isi hashtable, ternyata tidak ada yang cocok.
4. Untuk P=2, *hash value* dari substring BA cocok dengan isi hashtable yang pertama dari substring T (ini berarti nilai t=1) pada algoritma maka nilai k diset=s=2, kemudian bandingkan T[t+k] dengan P[p+k] yaitu T[3] dengan P[4], ternyata sama, maka naikkan nilai k menjadi 3, kemudian lakukan pengecekan T[4] dengan P[5], dan seterusnya hingga 3 kondisi berhenti berikut ditemui :
   a. jika token yang tidak sama ditemui.
   b. jika end *of list* dicapai.
   c. jika ditemukan token yang sudah ditandai.

   akhirnya ditemukan substring yang sama pada teks pola dan teks string yaitu BANAN dengan k=5

String Teks

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

String Pola

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

5. Lalu cek apakah k>2*s? ya, maka kembalikan k dan keluar dari fungsi ini untuk menuju Top Level RKRGST, Pada Top Level RKRGST Lmax=nilai yang dikembalikan oleh *scanpattern* yaitu 5, cek apakah Lmax>2*s, Ya, maka s=Lmax=5, lalu kembali ke repeat yaitu memanggil *scanpattern*, namun dengan nilai s yang baru dan kosongkan list-max-match
6. Kembali ke *Scanpattern* dengan s=5 kemudian buat hash table untuk semua substring t dengan panjang 5, yaitu : BANAN,ANANA,NANAR, dst… kemudian simpan di hashtable, untuk mempermudah contoh, kita umpamakan terlebih dahulu bahwa nilai hash sama dengan nilai substring nya.
7. Untuk pertama kali pada pengecekan P, dikarenakan setiap karakter pada string pola tidak terdapat mark, maka substring dapat langsung dibentuk dengan nilai s yang baru yaitu 5, Setelah di-hash dapat langsung dicocokkan dengan isi hashtable.
8. P=1, *hash value* substring PBANA di-hash lalu dicocokkan dengan semua isi hashtable, hasilnya tidak ada yang cocok.
9. P=2, *hash value* BANAN cocok dengan isi hashtable yang pertama (berarti nilai t=1),maka nilai k akan diset=s=5

String Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

String Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

10. Kemudian bandingkan T[t+k] dengan P[p+k] yaitu T[6] dengan P[7], ternyata tidak sama, maka keluar dari while lalu cek apakah k>2*s? tidak, maka

menuju ke else, yaitu menambahkan pola yang cocok, pada algoritma yang disimpan adalah indeksnya bukan stringnya dengan format (p,t,s).

11. Pada kasus lain Jika k > 2*s, maka langsung keluar dan kembalikan nilai k. lalu *scanpattern* dengan nilai s yang baru, nilai s yang baru adalah nilai kembalian dari k.
12. Untuk menambahkan pola yang cocok kedalam list, harus dicek terlebih dahulu apakah list ada isinya, karena jika ada, maka insert new item mesti melihat panjang k, list harus terurut secara descending berdasarkan panjang substring (k).
13. Isi list saat ini : (2,1,5), pada algoritma penulisan (p,t,k) artinya posisi pada pola berada pada karakter ke-2, posisi pada teks berada pada karakter ke-1, dengan panjang 5. sehingga didapatkan string BANAN, nantinya jika masuk ke fase markstring, pada pola yang akan di-mark adalah karakter ke-2 sepanjang 5, pada teks karakter ke-1 sepanjang 5, kemudian *hash value* substring BANAN dicocokkan kembali dengan elemen lain di hashtable, ditemukan ada yang sama yaitu pada elemen ke 40, maka nilai k diset=5, bandingkan T[t+k] dengan P[p+k] yaitu T[45] dengan P[7], ternyata tidak sama, maka keluar dari while, lalu cek apakah k>2*s? tidak, maka menuju ke else, yaitu menambahkan pola yang cocok , pada algoritma yang disimpan adalah indeksnya dengan format (p,t,s) bukan stringnya.
14. Saat ini pada *list-of-maximal-match* telah ada 1 pola, untuk itu dalam pengisian pola yang beru harus melihat panjang dari k, jika panjang dari k pada pola yang akan di isi kedalam list-of-maximal-match maka pola tersebut harus di insert keposisi terakhir atau dengan kata lain *list-of-maximal-match* harus terurut secara Descending berdasarkan nilai k, sehingga list-of-maximal-match saat ini berisi (2,1,5), (2,40,5).
15. Untuk sebuah substring di dalam string pola, akan dicari kesamaannya dengan semua elemen *hashtable*, karena mungkin saja akan ada lebih dari 1 substring di teks pola yang cocok dengan substring di string teks.
16. Melanjutkan *scanpattern*, *hash value* BANAN dicocokkan kembali dengan elemen lain di *hashtable*, ditemukan ada yang sama yaitu pada elemen ke 46,

maka nilai k diset = 5, bandingkan T[t+k] dengan P[p+k] yaitu T[51] dengan P[7], ternyata sama, maka naikkan k menjadi 6, lalu cocokkan P[8] dengan T[52], ternyata tidak ada T[52], atau teks sudah habis, maka keluar dari while, lalu cek apakah k>2*s? tidak, maka menuju ke else, yaitu menambahkan pola yang cocok, list-of-maximal-match harus terurut secara descending berdasarkan panjang substring (k), sehingga isi list menjadi : (2,46,6), (2,1,5), (2,40,5).

17. P=3, *hash value* substring ANANAT dihash lalu dicocokan dengan semua isi *hashtable*, dan ditemukan kecocokan pada elemen ke-47, maka k=s=5

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

bandingkan T[t+k] dengan P[p+k] yaitu T[47+5] dengan P[3+5], ternyata tidak ada T[52], atau teks sudah habis, maka keluar dari while, lalu cek apakah k>2*s? tidak, maka menuju ke else, yaitu menambahkan pola yang cocok, list harus terurut secara descending berdasarkan panjang substring (k), isi *list-of-maximal-match* menjadi : (2,46,6), (2,1,5), (2,40,5),(3,47,5)

18. P=4, *hash value* substring NANTB, tidak ada yang cocok pada hashtable, serta P=5,6,7 tidak ada yang cocok.
19. P=8, *hash value* substring BANAN, di-hash dan dicocokkan dengan hash-table, terdapat kesamaan pada elemen ke-1, maka k=s=5

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

bandingkan T[t+k] dengan P[p+k] yaitu T[6] dengan P[13], ternyata sama, maka naikkan k, kemudian cocokkan T[7] dengan P[14], ternyata tidak sama, selanjutnya keluar dari while, lalu cek apakah k>2*s? tidak, maka menuju ke else, yaitu menambahkan pola yang cocok, secara descending, sehingga isi *list-of-maximal-match* menjadi: (2,46,6),(8,1,6), (2,1,5), (2,40,5),(3,47,5)

tetap di P=8, *hash value* substring BANAN cocok dengan hashtable di elemen ke-40, set k=s=5, bandingkan T[t+k] dengan P[p+k] yaitu T[45] dengan P[13], ternyata sama, maka naikkan k menjadi 6, T[46] dg P[14], ternyata tidak sama, maka keluar dari while, lalu cek apakah k>2*s? tidak,

maka menuju ke else, yaitu menambahkan pola yang cocok, diurutkan secara descending, sehingga *list-of-maximal-match* menjadi: (2,46,6), (8,1,6), (8,40,6), (2,1,5), (2,40,5),(3,47,5).

Hash value substring BANAN dicocokkan kembali dengan elemen lain di hash table, ditemukan ada yang sama yaitu pada elemen ke 46, maka nilai k diset = 5, bandingkan T[t+k] dengan P[p+k] yaitu T[51] dengan P[13], ternyata tidak sama, maka keluar dari while, lalu cek apakah k>2*s?tidak maka menuju ke else, yaitu menambahkan pola yang cocok, diurutkan secara descending, sehingga list-of-maximal-match menjadi : (2,46,6), (8,1,6), (8,40,6), (2,1,5), (2,40,5),(3,47,5), (8,47,5).

20. P=9, *hash value* substring ANANA, di-hash, dan dicocokkan dengan elemen hashtable, didapat kecocokan dengan elemen ke-2, maka set k=s=5

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

bandingkan T[t+k] dengan P[p+k] yaitu T[7] dengan P[14], ternyata tidak sama, maka keluar dari while, lalu cek apakah k>2*s? tidak, maka menuju ke else, yaitu menambahkan pola yang cocok, diurutkan secara descending, sehingga *list-of-maximal-match* menjadi : (2,46,6), (8,1,6) ,(8,40,6), (2,1,5), (2,40,5), (3,47,5), (8,47,5), (9,2,5).

Hash value substring ANANA dicocokkan kembali dengan elemen lain di hash table, terdapat kesamaan dengan elemen hashtable yang ke-41 dan 47, dengan cara yang sama maka diperoleh isi *list-of-maximal-match* : (2,46,6), (8,1,6), (8,40,6), (2,1,5), (2,40,5),(3,47,5), (8,47,5), (9,2,5), (9,41,5)

21. P=10, *hash value* substring NANAM, tidak ada yang sama dengan hashtable serta p=11, 12, 13, 14, 15, tidak ada yang sama dengan hashtable
22. P=16 NARAM, dihash dan sama dengan elemen ke-5, set k=s=5, bandingkan T[t+k] dengan P[p+k] yaitu T[10] dengan P[21], ternyata sama, maka naikkan k mejadi 6, bandingkan T[11] dengan P[22], P[22] tdk ada, maka keluar dari while, lalu cek apakah k>2*s? tidak, maka menuju ke else, yaitu menambahkan pola yang cocok, , diurutkan secara *descending*. sehingga *list-of-maximal-match* menjadi : (2,46,6), (8,1,6) ,(8,40,6), (16,5,6), (2,1,5),

(2,40,5),(3,47,5), (8,47,5), (9,2,5), (9,41,5). *Hash value* substring NARAM dicocokkan kembali dengan elemen lain di hashtable sampai akhir pengecekan tidak ada yang sama.

23. P=17, *hash value* substring ARAMA, dengan cara yang sama menghasilkan *list-of-maximal-match* menjadi : (2,46,6), (8,1,6) ,(8,40,6), (16,5,6), (2,1,5), (2,40,5),(3,47,5), (8,47,5), (9,2,5), (9,41,5), (17,6,5)
24. P=18 ke atas sudah tidak memungkinkan karena panjangnya tidak mencapai 5 (panjang s).
25. Tahap berikutnya keluar dari *scanpattern* dan kembalikan nilai k terbesar (bisa diambil dari anggota pertama list), yaitu 6.
26. Kembali ke Top Level Algoritma RKR-GST, Lmax=hasil *scanpattern* tertinggi yaitu 6, jika Lmax > 2*s, tidak. Maka menuju ke else panggil markstring.
27. Pada fase markstring, dalam pembentukan sebuah tile diambil dari *list-of-maximal-match* yang telah diurutkan secara descending, saat ini telah terdapat *list-of-maximal-match* : (2,46,6), (8,1,6), (8,40,6), (16,5,6), (2,1,5), (2,40,5), (3,47,5), (8,47,5), (9,2,5), (9,41,5), (17,6,5).
28. Untuk maximal-match (2,46,6) maka T[46,..,46+6-1] dan P[2,..2+6-1] dan belum tertandai (*non mark*), maka lakukan pencocokkan pada setiap karakternya (jika pada fase *scanpattern* yang dicocokkan adalah nilai hash-nya, maka belum tentu nilai hash-nya sama, dan semua karakter di dalamnya juga sama, namun kemungkinan besar string itu sama). Setelah string tertandai maka length_of_token_tiled=6 dan hapus (2,46,6) dari *list-of-maximal-match*. sehingga string yang dimark saat ini menjadi :

Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

29. *list-of-maximal-match* saat ini : (8,1,6) ,(8,40,6), (16,5,6), (2,1,5), (2,40,5), (3,47,5), (8,47,5), (9,2,5), (9,41,5), (17,6,5). jika T[1,..,1+6-1] dan P[8…..8+6-1] tidak terdapat mark, maka lakukan pencocokkan pada setiap string (jika pada fase *scanpattern* yang dicocokkan adalah nilai hash-nya, maka belum tentu nilai hash-nya sama, dan semua karakter di dalamnya juga sama, namun kemungkinan besar string-nya sama) sehingga menghasilkan length_of_token_tiled=12 dan hapus (8,1,6) dari list-of-maximal-match

Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

30. List-of-maximal-match saat ini : (8,40,6), (16,5,6), (2,1,5), (2,40,5),(3,47,5), (8,47,5), (9,2,5), (9,41,5), (17,6,5), jika T[40,..,40+6-1] dan P[8,..8+6-1] tidak di mark? Ada. Jika terdapat 1 saja yang telah ada mark dari salah satunya, maka pada algoritma menuju ke *else if* (L - panjang token yang di-mark pada p maupun t) >= s, maka masukkan sisa yang belum bertanda ke list-of-maximal-match, jika panjang token yang ditandai pada p dan t berbeda, maka panjang token yang ditandai harus diambil yang terpanjang, pada kasus ini T[40..45] belum ditandai, P[8..13] sudah ditandai semua, maka L - length of mark=6-6=0, sehingga tidak ada yang ditambahkan dan hapus (8,40,6) dari list-of-maximal-match

Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

31. list-of-maximal-match saat ini menjadi : (16,5,6), (2,1,5), (2,40,5),(3,47,5), (8,47,5), (9,2,5), (9,41,5), (17,6,5). Pada list-of-maximal-match (16,5,6) Setelah ditelusuri T[5..10] terdapat karakter yang telah ditandai sebanyak 2, dan pada P[16..21] semua karakter tidak ada yang tertandai, maka L-length of mark=6-2=4, dikarenakan 4 < s, maka tidak perlu ditambahkan ke list-of-maximal-match dan tidak ada penandaan di sini kemudian hapus (16,5,6) dari list-of-maximal-match
32. kasus lain seandainya saja, s tidak 5, misalnya s=2, maka L-marked=4, 4>s, sehingga sisanya mesti ditambahkan ke list-of-maximal-match (yang ditambahkan adalah bagian yang belum tertandai). Artinya isi list menjadi : (2,1,5), (2,40,5),(3,47,5), (8,47,5), (9,2,5), (9,41,5), (17,6,5), (18,7,4) dengan list baru inilah akan dilakukan penandaan kembali
33. kembali ke kasus sebelumnya dan lanjutkan penandaan, isi list-of-maximal-match saat ini: (2,1,5), (2,40,5),(3,47,5), (8,47,5), (9,2,5), (9,41,5), (17,6,5), untuk T[5..10] telah tertandai semua, dan P[12..6] telah tertandai semua, maka tidak dilakukan apapun dan hapus (2,1,5) dari list-of-maximal-match
34. Lakukan cara yang sama sampai list-of-maximal-match kosong. Sehingga didapatkan string yang telah ditandai menjadi seperti berikut

Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

35. Selanjutnya setelah list-of-maximal-match kosong program akan keluar dari fase markstring, selanjutnya jika s>2*min_match_length (jika 5>2*2), Jika ya, maka kembali ke repeat yaitu masuk kembali ke fase *scanpattern* dengan nilai s = (5 div 2) = 2, panggil *scanpattern* dengan s=2
36. karena saat ini telah terdapat *mark* pada karakter didalam string teks dan string pola, maka untuk pembentukan substring mulai memperhatikan *mark* string yang ada. Pada kondisi ini substring akan dibentuk jika panjang substring memenuhi panjang s dan tidak terdapat *mark* pada setiap string-nya.
37. Pembentukan hashtable dari teks dimulai dari indeks ke-7, yaitu substring RA, AM, MA, dst, sampai indeks ke 44 yaitu substring NA.

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

38. P=1, jika jarak ke next tile < s, maka tidak dapat dibentuk substring dikarenakan panjangnya tidak mencapai s. sehingga majukan ke akhir dari tile, yaitu P=14.
39. Untuk P=14, *hash value* substring MU, sama dengan elemen hashtable index ke-20 dan 34 dan dengan cara yang sama menghasilkan list-of-maximal-match (14,20,2) (14,34,2)

string Pola

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

40. P=15 tidak ada yang sama
41. p=16, hash value NA, sama dengan hash value substring 42 dan 44, dengan cara yang sama menghasilkan list-of-maximal-match : (14,20,2), (14,34,2), (16,42,2), (16,44,2).
42. P=18, hash value substring RA, sama dengan hash value elemen ke-7 hashtable, maka set nilai k=2, T[7+2] dibandingkan dengan P[18+2], ternyata sama, maka set nilai k=3, lanjutkan pengecekan T[7+3] = P[18+3], ternyata sama, maka set nilai k=4, dan keluar dari while karen posisi index P

telah berada diakhir dari string pola. Kemudian cek apakah k>2*s? tidak, maka tambahkan ke list-of-maximal-match, isi list : (18,7,4),(14,20,2) (14,34,2),(16,42,2), (16,44,2).

43. P=19, substring AM sama dengan elemen ke-8, maka set k = 2, lanjutkan pencocokan pada index selanjutnya yaitu T[8+2] dan P[19+2] ternyata sama maka set k=3, pada index selanjutnya tidak terdapat lagi kesamaan sehingga didapat max-match (19,8,3), isi list-of-maximal-match menjadi : (18,7,4), (19,8,3), (14,20,2), (14,34,2),(16,42,2), (16,44,2).
44. Diakhir dari substring P=20, isi list-of-maximal-match menjadi : (18,7,4),(19,8,3), (14,20,2) (14,34,2),(16,42,2), (16,44,2), (20,9,2).
45. Setelah tidak ada lagi substring yang dapat dibentuk maka keluar dari fase *scanpattern* menuju Top Level Algoritma RKRGST, dengan nilai kembalian k terpanjang yaitu 4, di Top Level Algoritma RKRGST, Lmax=4, Lmax > 2*s? tidak, maka menuju *else*, panggil markstring.

Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

46. List-of-maximal-match : (18,7,4), (19,8,3), (14,20,2), (14,34,2), (16,42,2), (16,44,2), (20,9,2). Untuk max-match (18,7,4) apakah semua elemen T[7..10] dan P[18..21] tak bertanda? Ya. Maka cocokkan setiap elemen satu per satu, jika terdapat kecocokan maka tandai, kemudian hapus (18,7,4) dari list

Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

Pola (p)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

47. isi list : (19,8,3), (14,20,2) (14,34,2),(16,42,2), (16,44,2), (20,9,2)

    Untuk max-match (19,8,3) apakah semua T[8,10] dan P[19..21] tak ditandai? tertandai, jumlah yang tak bertanda 0<s , maka tidak dilakukan apapun dan hapus (19,8,3)

48. isi list : (14,20,2), (14,34,2),(16,42,2), (16,44,2), (20,9,2)

    Untuk max-match (14,20,2) apakah semua T[20..21] dan P[14..15] tdk ditandai? ya, Maka cocokkan setiap elemen satu per satu, jika terdapat kecocokan maka tandai, kemudian hapus (14,20,2)

Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

49. isi list : (14,34,2), (16,42,2), (16,44,2), (20,9,2)

    Untuk max-match (14,34,2) apakah semua T[34..35] dan P[14..15] tdk ditandai? tertandai, P[14..15] telah tertandai semua, sehingga jumlah tak ditandai 0<s, maka hapus (14,34,2).

50. isi list : (16,42,2), (16,44,2), (20,9,2)

    Untuk max-match (16,42,2) apakah seluruh string T[42..43] dan P[16..17] tidak ditandai? Ya, Maka cocokkan setiap elemen satu per satu, jika terdapat kecocokan maka tandai, kemudian hapus (16,42,2).

Teks (T)

B	A	N	A	N	A	R	A	M	A	A	K	U	S	U	K	A	K	A	M
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

U	T	A	P	I	A	P	A	K	A	H	K	A	M	U	J	U	G	A	B
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

A	N	A	N	A	B	A	N	A	N	T
41	42	43	44	45	46	47	48	49	50	51

Pola (P)

P	B	A	N	A	N	T	B	A	N	A	N	A	M	U	N	A	R	A	M	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

51. isi list : (16,44,2), (20,9,2)

    Untuk max-match (16,44,2) apakah semua T[44..45] dan P[16..17] tidak ditandai? tertandai, untuk P[16..17] jumlah yang tidak tertandai=0<s, maka hapus (16,44,2) dari list

52. isi list : (20,9,2)

    Untuk max-match (20,9,2) apakah semua T[9..10] dan P[20..21] tidak ditandai? semua telah ditandai, jumlah yang tdk ditandai=0<s, maka hapus (20,9,2)

53. Selanjutnya menuju ke Top Level Algoritma RKRGST

    Keluar dari fase markstring menuju statement berikutnya

    apakah s>2*min_match_length? Tidak, maka menuju ke elseif

    apakah s>min_match_length ? tidak, maka

    else stop