Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Team project ©2017
Dony Pratidana S. Hum | Bima Agus Setyawan S. IIP

BAB III

ANALISIS DAN PERANCANGAN

3.1 Analisis Masalah

Berdasarkan analisis yang dilakukan masalah yang terjadi dalam tindak plagiarisme kode program adalah adanya tindakan plagiat yang semakin mudah dilakukan dikalangan masyarakat khususnya di kalangan mahasiswa. Sedangkan, untuk mendeteksinya akan membutuhkan waktu yang lama bagi pengajar apabila jumlah program yang harus diperiksa cukup banyak.

Untuk mengatasi hal tersebut telah dilakukan banyak penelitian dan pembuatan aplikasi pendeteksi plagiarisme pada kode program. Salah satunya penelitian yang dilakukan oleh Andreas Arifianto (Arifianto, 2011) yang menghasilkan aplikasi pendeteksi plagiarisme dengan menggunakan algoritma *Levenshtein Distance* sebagai dasar untuk melakukan pencocokan *string* dan proses diawali dengan melakukan teknik *preprocessing.*

Adapun algoritma *Levenshtein Distance* sering digunakan terutama dalam konteks *string*. Pada algoritma ini jarak antara dua *string* ditentukan oleh jumlah minimal operasi yang dibutuhkan untuk mengubah satu *string* ke *string* lainnya. Operasi yang dapat dilakukan adalah *substitution*, *insertion*, dan *deletion*. Algoritma ini cukup baik digunakan namun bukan menjadi salah satu algoritma yang diandalkan dalam pencocokan *string.* Selain itu, teknik *preprocessing* yang dilakukan oleh peneliti terdahulu hanya dapat memformat variabel, *conditional*

UNIVERSITAS MULTIMEDIA NUSANTARA

statement, *looping* dan tanda – tanda baca. Hal ini menyebabkan aplikasi bekerja dengan batasan yang cukup kecil.

3.2 Usulan Pemecahan Masalah

Berdasarkan masalah yang ditemukan tersebut, ditawarkan solusi berupa pembuatan sebuah aplikasi pendeteksi plagiarism dengan kemampuan:

1. Mendeteksi dengan cara membandingkan antara dua *file* kode program dalam waktu yang sama berdasarkan karakter dan baris kemudian ditampilkan hasilnya dalam bentuk persentase kecenderungan plagiarisme yang terjadi.
2. Melakukan modifikasi pada teknik *preprocessing* yang ada pada aplikasi yang telah dibuat sebelumnya sehingga dapat memiliki batasan yang lebih luas dan dapat mendeteksi setiap unsur yang terdapat dalam bahasa pemrograman C.
3. Menghasilkan persentase kecenderungan plagiarisme antara setiap dua *file* kode program yang terdapat di dalam *folder* yang dipilih dengan algoritma *Boyer Moore* (metode pencocokan berdasarkan karakter) dan algoritma *Brute Force* (metode pencocokan berdasarkan baris),

3.3 Spesifikasi Umum Kebutuhan Sistem

Spesifikasi umum kebutuhan sistem ini menjelaskan mengenai masukan dan keluaran dari sistem yang dibuat.

UNIVERSITAS MULTIMEDIA NUSANTARA

3.3.1 Masukkan dan Keluaran Sistem

Masukkan yang dibutuhkan sistem berupa:

1. Lokasi penyimpanan dari dua *file* kode program yang akan dibandingkan.
2. Lokasi penyimpanan dari satu *folder* yang berisi sedikitnya dua *file* kode program dengan format .c dan .cpp.

Keluaran dari sistem berupa:

1. Nilai persentase kecenderungan plagiarisme antara dua buah program berdasarkan karakter dan baris.
2. Sebuah *file* dengan nama "CompareResult" dan berformat .txt yang berisi hasil pencocokan setiap dua buah *file* program yang ada dalam *folder* yang dipilih untuk dibandingkan isinya.

3.4 Perancangan Sistem

Tahapan perancangan dari sistem yang dibuat yaitu:

1. Tahap *preprocessing* sebagai tahap awal sebelum dilakukan pembandingan kode program.
2. Tahap penghitungan persentase kecenderungan plagiarisme dengan algoritma *Boyer Moore* berdasarkan karakter.
3. Tahap penghitungan persentase kecenderungan plagiarisme dengan algoritma *Brute Force* berdasarkan baris.
4. Tahap pembandingan antara dua *file* kode program.
5. Tahap pembandingan setiap dua *file* di dalam satu *folder*.
6. Adanya petunjuk penggunaan dari sistem

3.4.1 Proses sistem

A. Tahap *Preprocessing*

Pada tahap *preprocessing*, *file* kode program bahasa C yang akan dibandingkan akan diubah menjadi suatu bentuk yang sama dengan tujuan agar nilai persentase kecenderungan plagiarisme yang dihasilkan dapat lebih akurat.

Teknik *preprocessing* akan mengubah data mentah (*file* kode program awal) menjadi sebuah data yang siap dibandingkan sebagai *string* biasa. Teknik ini juga sangat berguna dalam pendeteksian plagiarisme *file* kode program. Hal ini disebabkan karena dalam proses plagiarisme kode program ada beberapa bagian yang dapat diubah tanpa mengubah makna kode program secara keseluruhan.

Preprocessing yang dilakukan adalah dengan memodifikasi *preprocessing* yang sudah ada dengan menambahkan batasan baru. Hal ini dilakukan agar aplikasi dapat mendeteksi kecenderungan plagiarisme pada kode program yang mengandung semua elemen dalam bahasa C termasuk *array*, *struct*, *file* dan *function.*

Adapun proses *preprocessing* yang terjadi di dalam aplikasi pendeteksi plagiarisme kode program adalah sebagai berikut:

1. Menghapus seluruh komentar yang terdapat di dalam kode program, baik diawali // maupun /* .. */.
2. Menghapus seluruh *header* dalam kode program C.
3. Menghapus seluruh isi *string* (teks yang berada di antara tanda "…").
4. Menambahkan karakter '{' dan '}' pada setiap bentuk *if*, *while*, dan *do…while*, *struct* satu baris.

5. Menyeragamkan/menukar posisi *conditional statement* lebih dari (>) dan lebih dari atau sama dengan (>=) menjadi bentuk kurang dari (<) dan kurang dari atau sama dengan (<=).
6. Mengubah seluruh bentuk *for* menjadi bentuk *while*.
7. Menyeragamkan penamaan pada deklarasi variabel berdasarkan tipe data yang digunakan.
8. Menyeragamkan penamaan fungsi dan parameter fungsi berdasarkan tipe data parameter fungsi yang digunakan.
9. Menghapus seluruh spasi, beberapa tanda baca dan simbol, serta memformat struktur kode program.

Tabel 3.1 Tabel Contoh Hasil Sebelum dan Sesudah *Preprocessing*

Bentuk Sebelum *Preprocessing*	**Bentuk Setelah *Preprocessing***
`int column;`	`intvar-int;`

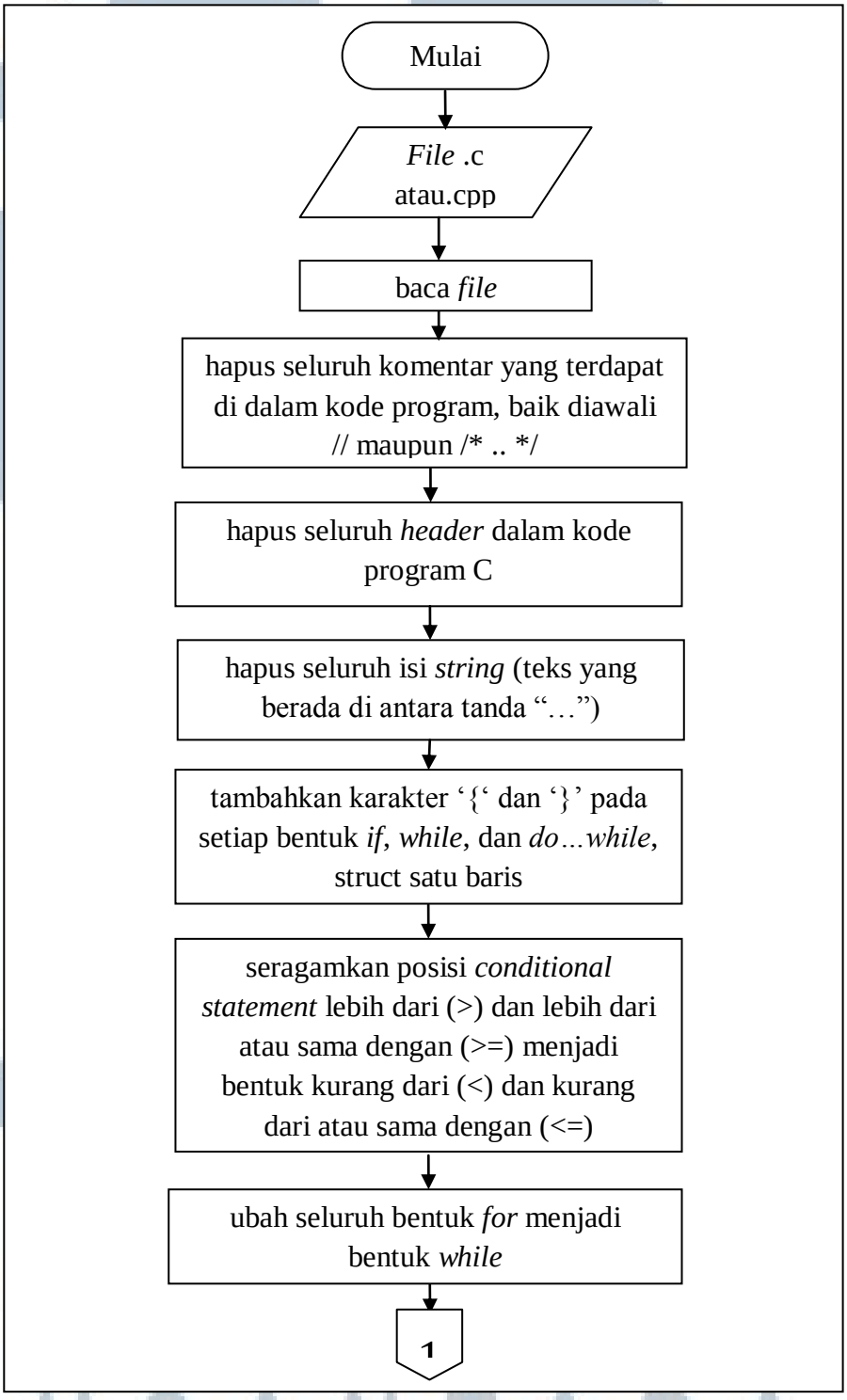
`if ( ( test && test ) != 0 ) {`
`klpt = 0;`
`break;`
`}`
	`ifparam-int&&var-int!=0`
	`var-int=0;`
	`break;`

`for ( i = 1; i >= 10; i++)`	`whilevar-int<=10`
`void Test(int data[], int data2)`	`voidfunc-functionintparameter-int[],intparameter-int`
`FILE *fp;`	`FILEvar-file;`
`int Array[10];`	`intvar-vararray[];`

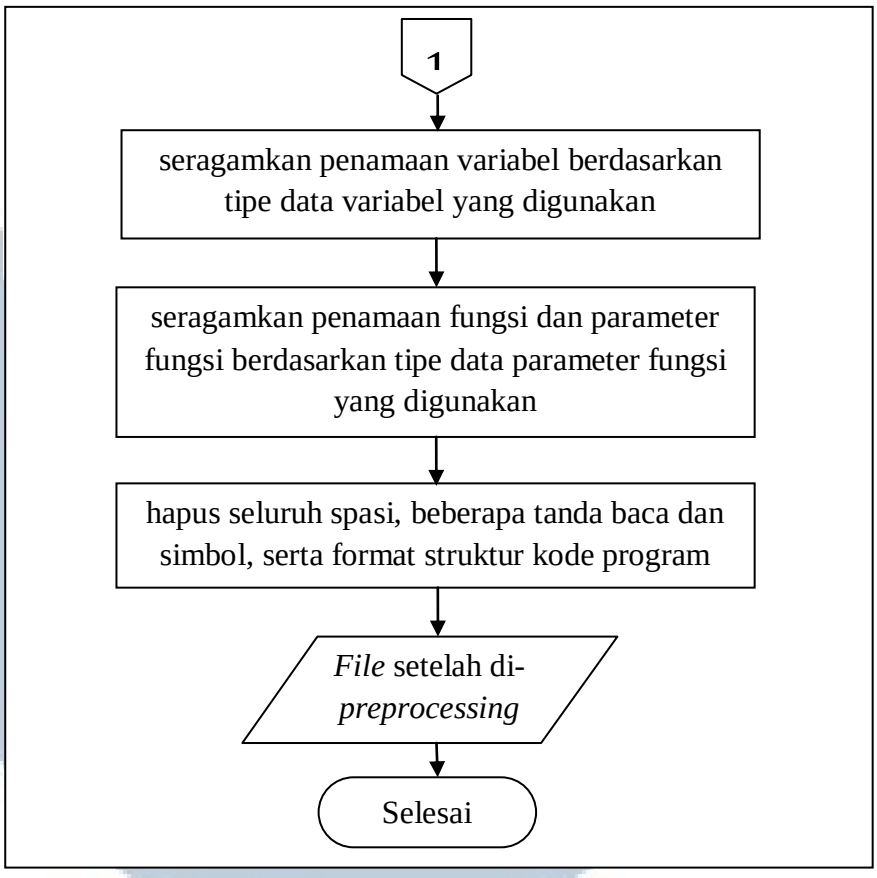
`struct Test {`
`int nama; /* nama mahasiswa */`
`char nim[ 10 ];//nim mahasiswa`
`};`
	`structTestintvar-intcharvar-char;`

UNIVERSITAS MULTIMEDIA NUSANTARA

Jika digambarkan dalam bentuk *flowchart*, maka tahapan-tahapan dari *preprocessing* yang terjadi dapat dilihat pada gambar berikut.



Gambar 3.1 *Flowchart Preprocessing*



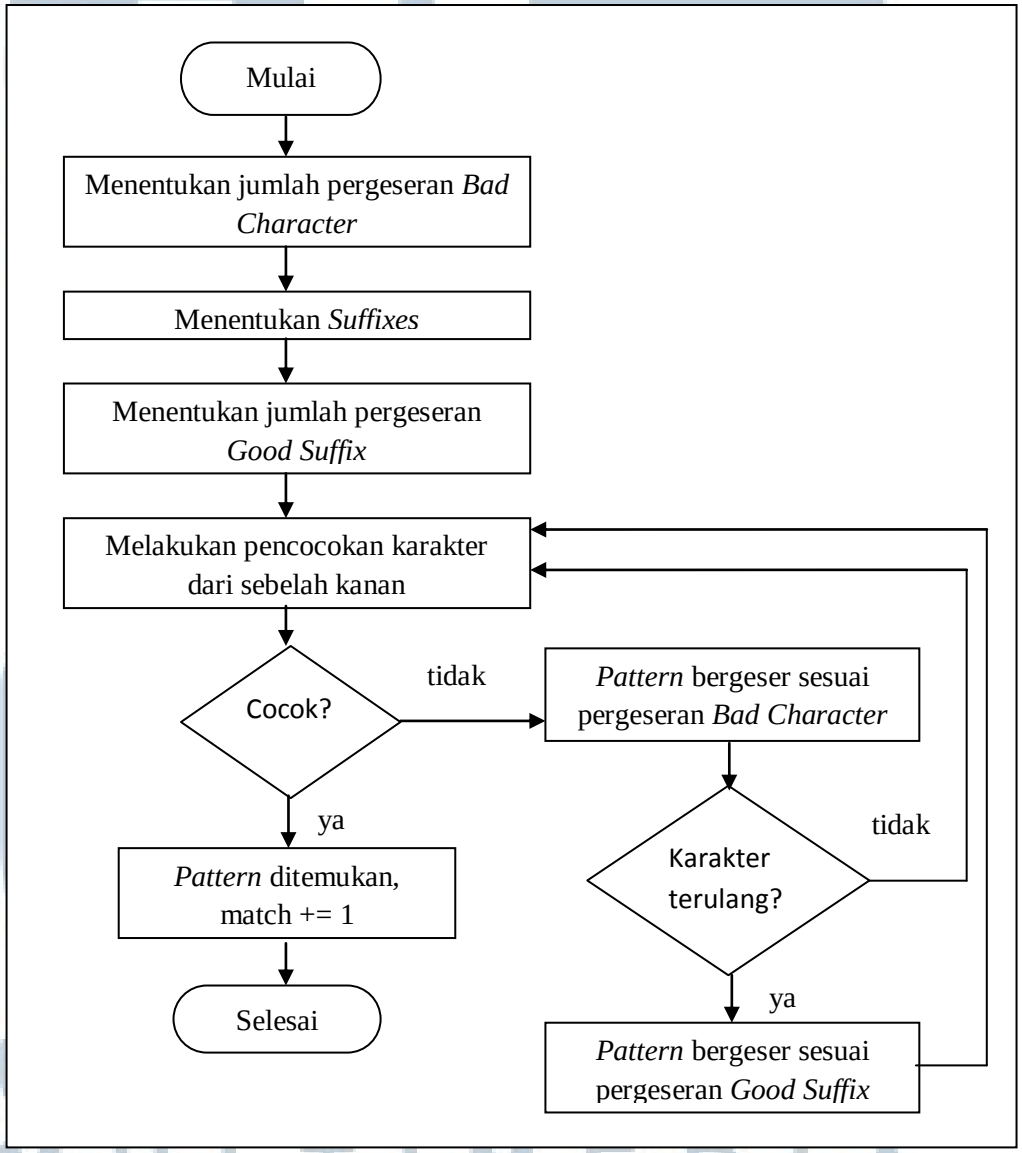
Gambar 3.1 *Flowchart Preprocessing* (Lanjutan)

B. Tahap Penghitungan Nilai Persentase Kecenderungan Plagiarisme Dengan Algoritma *Boyer Moore*

Setelah *preprocessing* dilakukan, kode program pertama dan kedua akan menjadi *file* pertama dan kedua. Proses selanjutnya yang dilakukan adalah perhitungan nilai untuk menentukan persentase kecenderungan plagiarisme antara *file* pertama dan kedua yang dibandingkan. Metode pertama yang digunakan dalam penilaian adalah dengan menggunakan algoritma *Boyer Moore*.

Dalam penghitungan nilai persentase plagiarisme dengan algoritma *Boyer Moore* setiap string pada *file* pertama dan kedua akan dicocokkan dengan algoritma ini. *String* pada kedua *file* tersebut akan berperan sebagai *pattern* dan

text. *File pattern* akan mencocokan setiap polanya pada setiap *string* pada *file text*. Jika ditemukan yang cocok maka *counter* jumlah kecocokan yang ditemukan akan bertambah satu. Yang bertindak sebagai *pattern* ialah *file* dengan jumlah *string* yang dibandingkan yang terkecil, sedangkan *file* yang menjadi *text* adalah sebaliknya.



Gambar 3.2 *Flowchart* Pembandingan Dengan Algoritma *Boyer Moore*

MULTIMEDIA NUSANTARA

Apabila setiap *string* pada *file pattern* sudah dicocokkan dan telah ditemukan jumlah kesamaan yang terjadi. Maka selanjutnya dilakukan penghitungan persentase dari kecenderungan plagiarisme antar 2 buah kode program yang dibandingkan dengan dengan teknik statistik sederhana (Warsito, 1992, hal.59):

$$\text{Persentase Berdasarkan Karakter} = \frac{\text{Jumlah pattern yang sama} * 100\%}{\text{nPattern}}$$

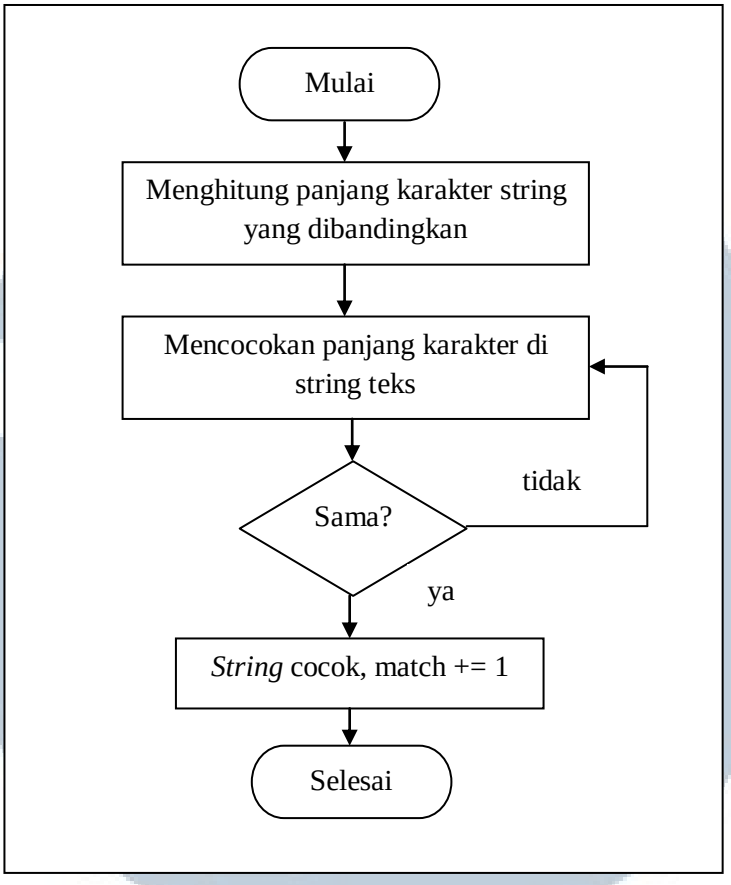
**nPattern = jumlah string yang dibandingkan pada file pattern*

C. Tahap Penghitungan Nilai Persentase Kecenderungan Plagiarisme Dengan Algoritma *Brute Force*

Selain menggunakan algoritma *Boyer Moore*, sistem juga melakukan penghitungan nilai persentase kecenderungan plagiarisme *file* kode program dengan menggunakan algoritma *Brute Force*.

Pada algoritma *Brute Force*, setiap baris pada *file* yang memiliki jumlah baris terpendek akan dicocokan dengan seluruh baris pada *file* lainnya. Jika ditemukan kecocokan, maka indikasi kecocokan akan bertambah dan baris tersebut akan dihapus. Hal ini dilakukan agar, jika terdapat 2 baris kode program yang identik pada program 1, sedangkan hanya terdapat 1 baris kode program yang identik dengan kedua baris kode pada program 2, maka indikasi kecocokan hanya akan bertambah 1.

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.3 *Flowchart* Pembandingan Dengan Algoritma *Brute Force*

Lalu untuk menghasilkan bobot persentase kecenderungan plagiarisme berdasarkan algoritma *Brute Force* dapat dilakukan dengan teknik statistik sederhana (Warsito, 1992, hal.59):

$$\text{Persentase Berdasarkan Baris} = \frac{\text{Jumlah baris yang sama} * 100\%}{\text{max_line}}$$

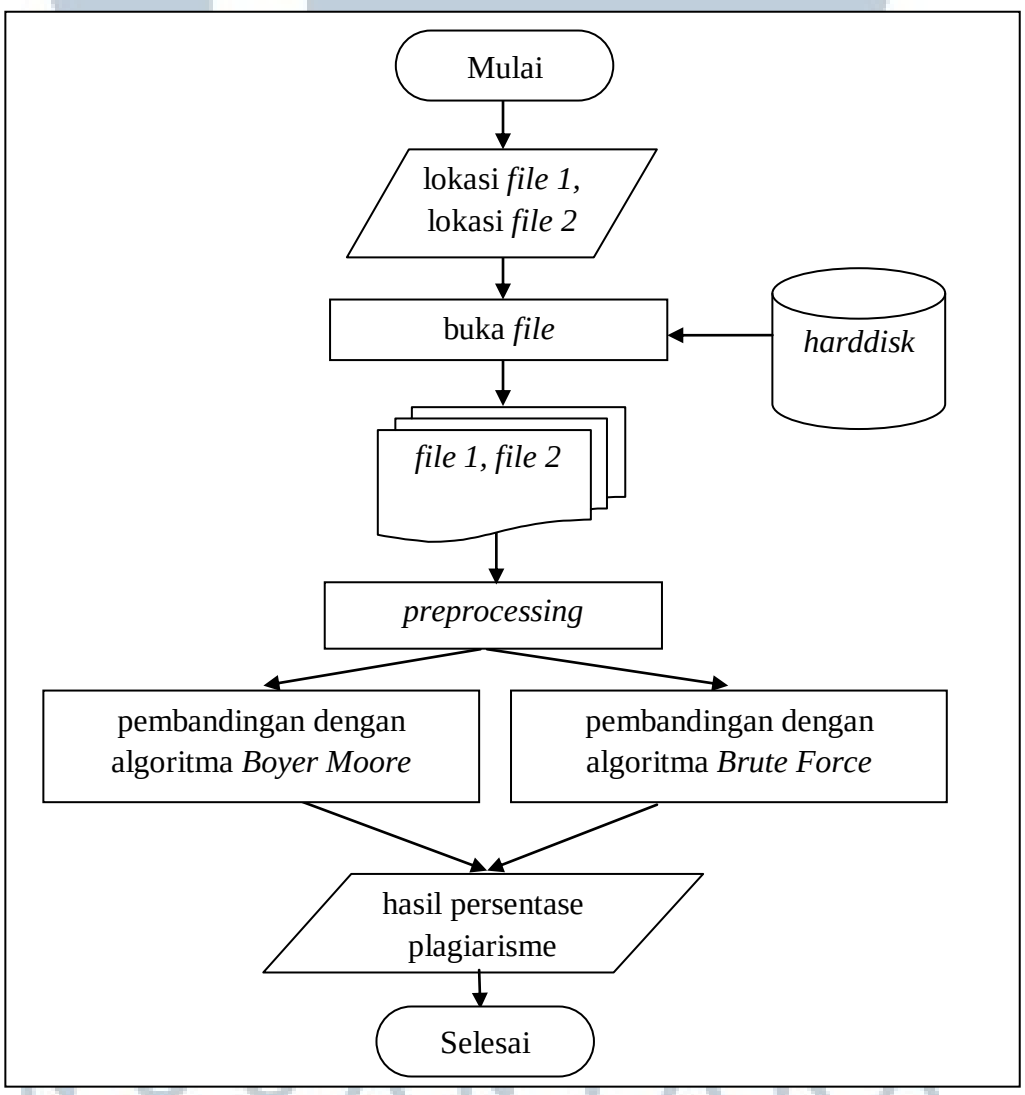
**max_line = jumlah baris terpanjang antara file pertama dan file kedua*

Kompleksitas dari algoritma *Brute Force* sendiri adalah O(m, n) untuk worsth case, dimana 'm' adalah jumlah baris dari program 1 dan 'n' adalah jumlah baris dari program 2. Sedangkan untuk *best case* adalah O(x), dimana x adalah jumlah baris terpanjang antara program 1 dan program 2.

D. Tahap Pembandingan Antara Dua *File* Kode Program

Tahapan berikutnya adalah menggabungkan ketiga proses yang ada di tahap sebelumnya sehingga dapat menghasilkan persentase kecenderungan plagiarisme pada kode program. Untuk itu maka dirancang sebuah fitur dimana *user* dapat membandingkan dua *file* kode program dalam satu waktu dan dapat melihat hasil persentase plagiarism yang terjadi pada kode program yang dibandingkan.

Adapun alur dari proses pembandingan 2 *file* kode program dapat dilihat pada gambar *flowchart* berikut.

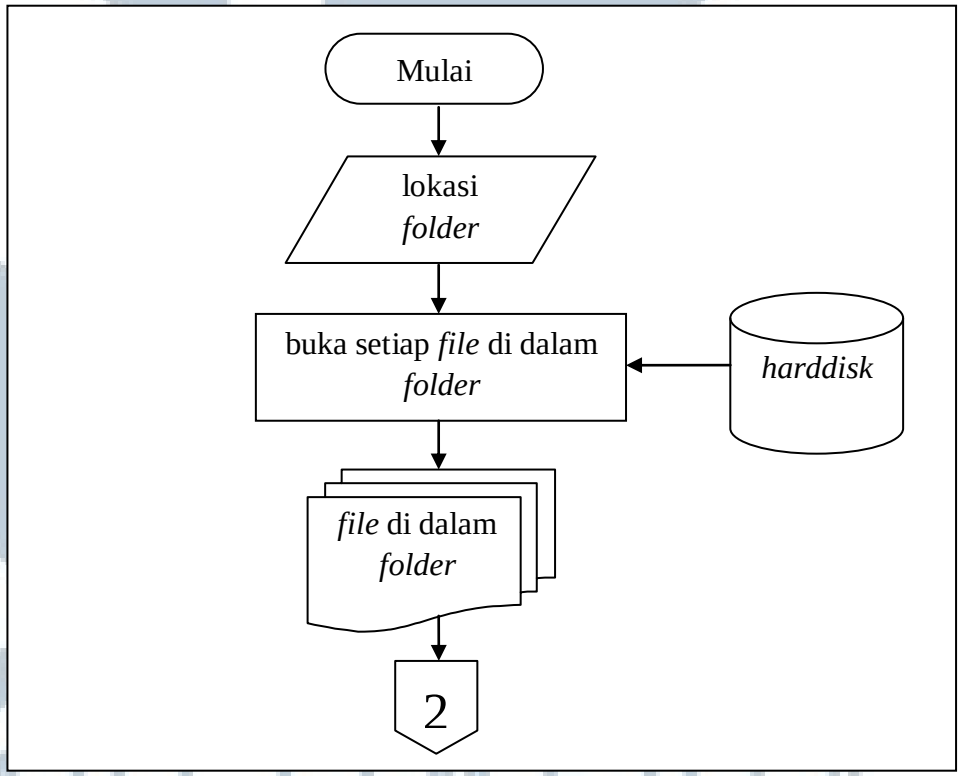


Gambar 3.4 *Flowchart* Proses Perbandingan Antara Dua *File* Kode Program

E. Tahap Pembandingan Antara *File* Kode Program di Dalam *Folder*

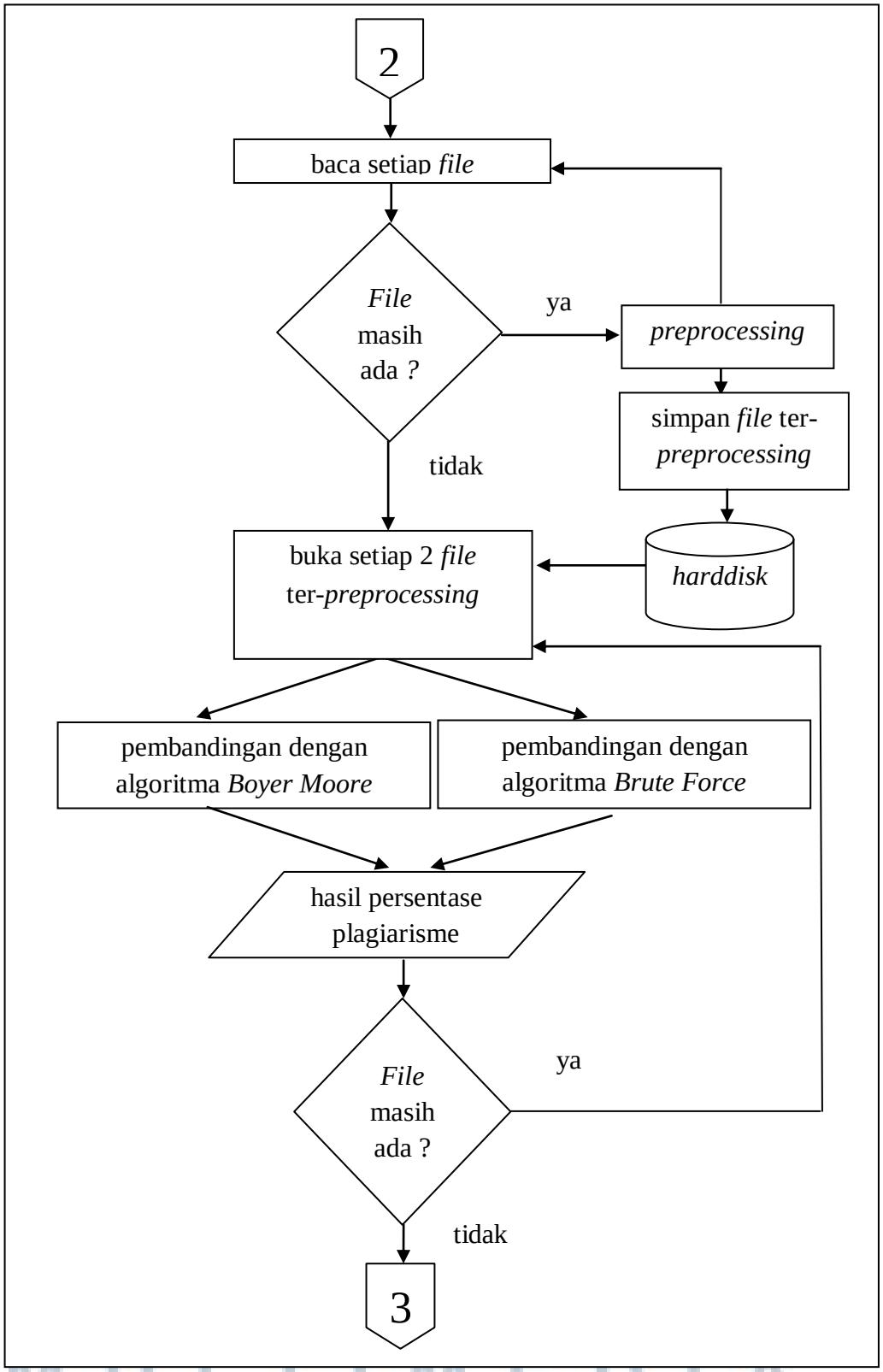
Setelah merancang fitur perbandingan antara dua *file* kode program, penulis dirancang sebuah fitur lagi untuk melakukan perbandingan apabila *file* yang ingin dibandingkan dalam jumlah banyak. Untuk itu dirancang fitur untuk membandingkan seluruh *file* kode program yang diletakkan di dalam satu *folder* tertentu. Tidak berbeda dari fitur yang dirancang sebelumnya dalam fitur ini *user* dapat melihat persentase kecenderungan plagiarisme antara setiap dua *file* kode program yang terdapat di dalam *folder* tersebut.

Adapun alur dari proses perbandingan kode program di dalam *folder* dapat dilihat pada gambar *flowchart* berikut:



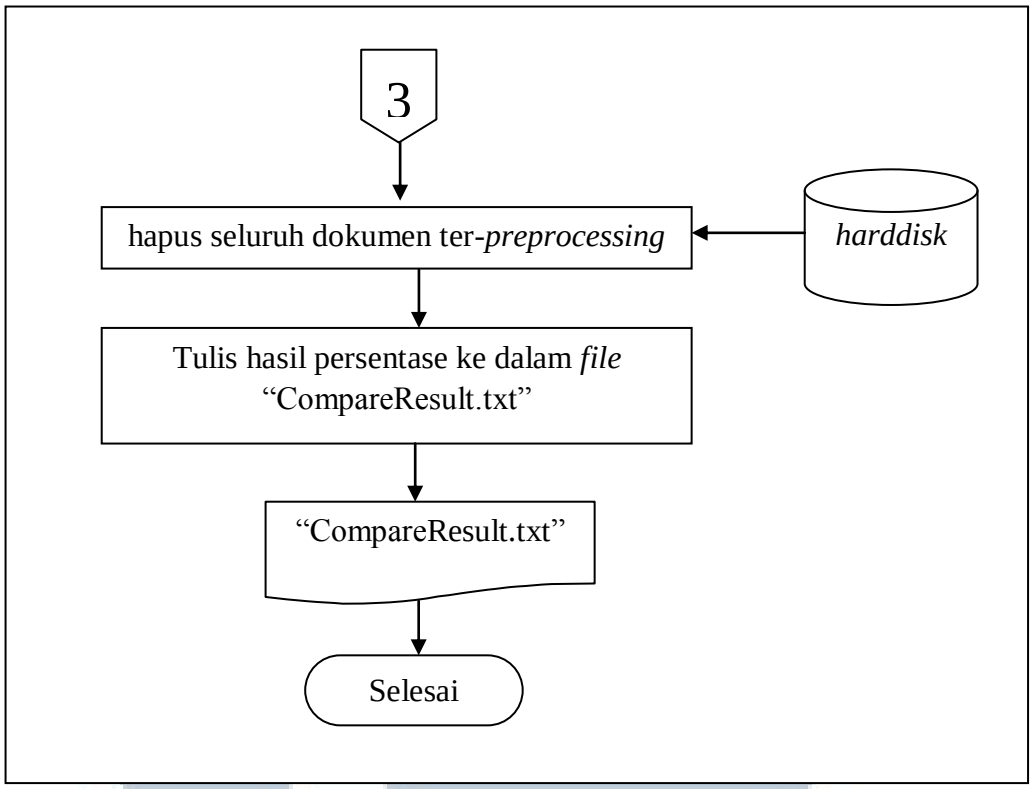
Gambar 3.5 *Flowchart* Proses Perbandingan Antara *File* Kode Program di Dalam *Folder*

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.5 *Flowchart* Proses Perbandingan Antara *File* Kode Program di Dalam *Folder* (Lanjutan)

MULTIMEDIA NUSANTARA



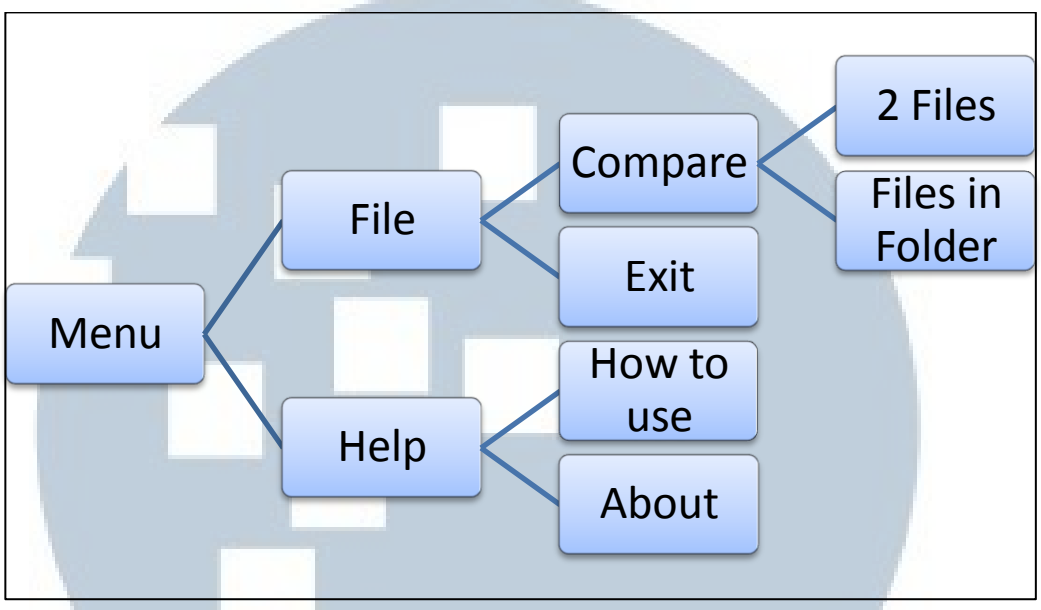
Gambar 3.5 *Flowchart* Proses Perbandingan Antara *File* Kode Program di Dalam *Folder* (Lanjutan)

F. Petunjuk Penggunaan dan Informasi Sistem

Setelah kedua fitur utama selesai dirancang, tahap yang terakhir adalah merancang fitur untuk memberikan petunjuk penggunaan dan informasi mengenai aplikasi yang dibangun. Bagian ini akan berfungsi sebagai panduan bagi *user* dalam menggunakan aplikasi. Panduan ini akan menjelaskan informasi secara umum mengenai aplikasi, dan urutan langkah cara pemakaian dari fitur yang disediakan.

3.4.2 Struktur Hirarki Menu

Struktur hirarki menu dari aplikasi pendeteksi plagiarisme yang dibuat dapat digambarkan sebagai berikut:



Gambar 3.6 Struktur Hirarki Menu

Dari menu utama, aplikasi dibagi menjadi 2 bagian besar, yakni "*Compare*" dan "*Help*". Menu "*Compare*" dibagi lagi menjadi 2 bagian, yakni "*2 Files*" dan "*Files in Folder*". Halaman "*2 Files*" berfungsi untuk membandingkan 2 buah *file* kode program. Sedangkan halaman "*Files in Folder*" berfungsi untuk membandingkan seluruh *file* kode program yang terdapat dalam satu *folder* tertentu yang dipilih user dari halaman ini.

Pada menu "*Help*", terdapat navigasi untuk mengakses halaman "*How To Use*" dan "About". Halaman "*How To Use*" berfungsi untuk menjelaskan fungsi dan cara penggunaan dari aplikasi pendeteksi plagiarisme kepada user. Sedangkan halaman "*About*" berisi informasi secara umum tentang aplikasi yang dibuat.

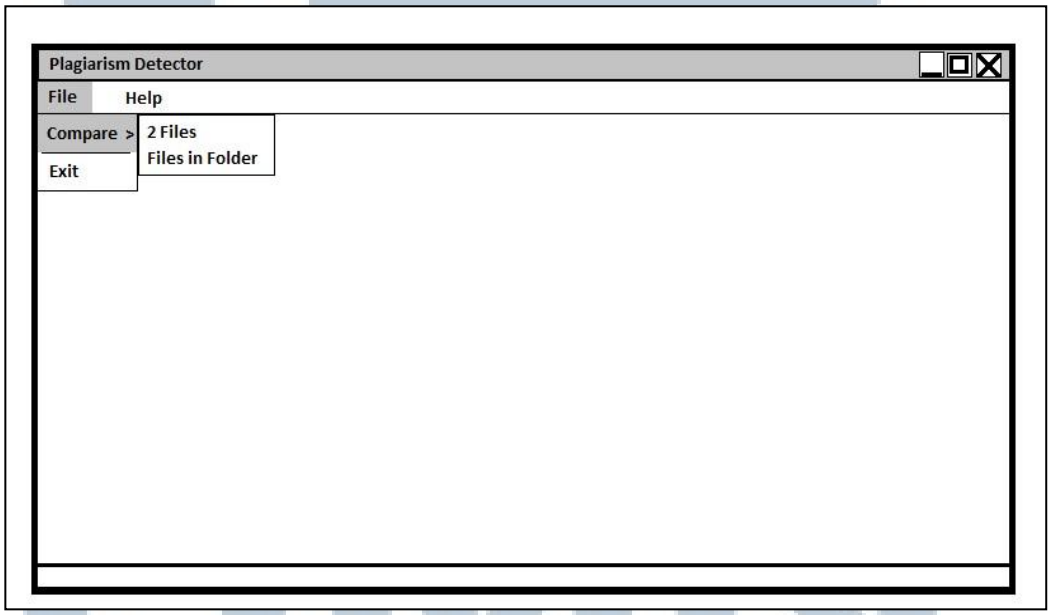
UNIVERSITAS MULTIMEDIA NUSANTARA

3.4.3 Tampilan Antarmuka

Desain tampilan antarmuka yang akan dibuat ke dalam aplikasi antara lain:

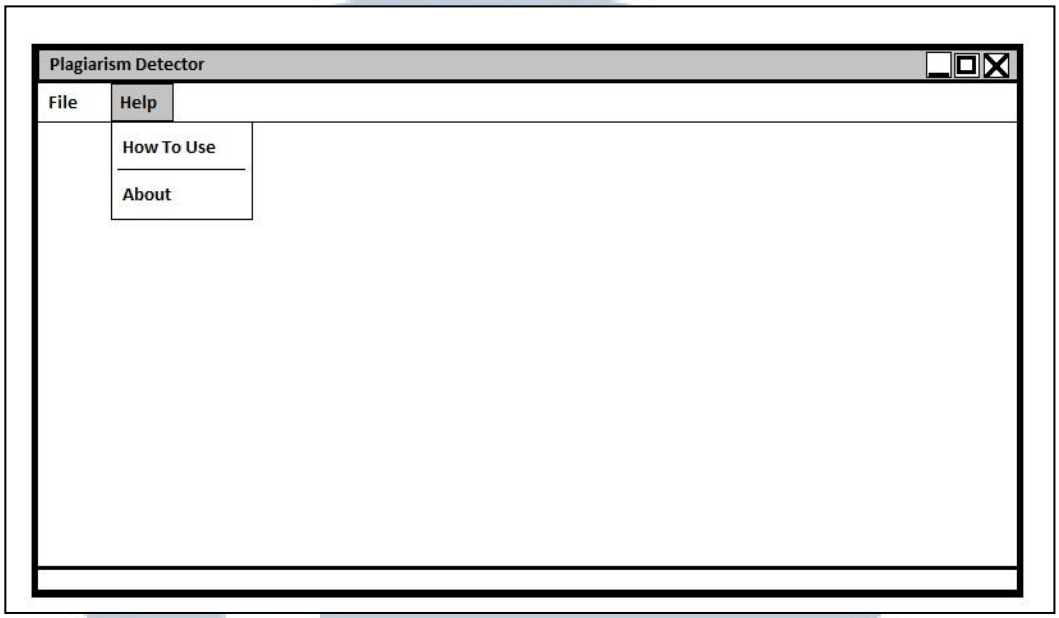
1. Halaman Utama

   Halaman utama berfungsi sebagai halaman awal saat aplikasi dibuka dan sebagai pusat dari pemilihan fitur yang disediakan. Pada halaman ini terdapat menu utama "*File*" dan "Help", dimana menu "*File*" terdiri dari submenu "*Compare*" dan "*Exit*" serta "*Help*" yang terdiri dari submenu "*How To Use*" dan "*About*". Juanita sari misal



Gambar 3.7 Desain Tampilan Antarmuka Halaman Utama Menu "*Compare*"

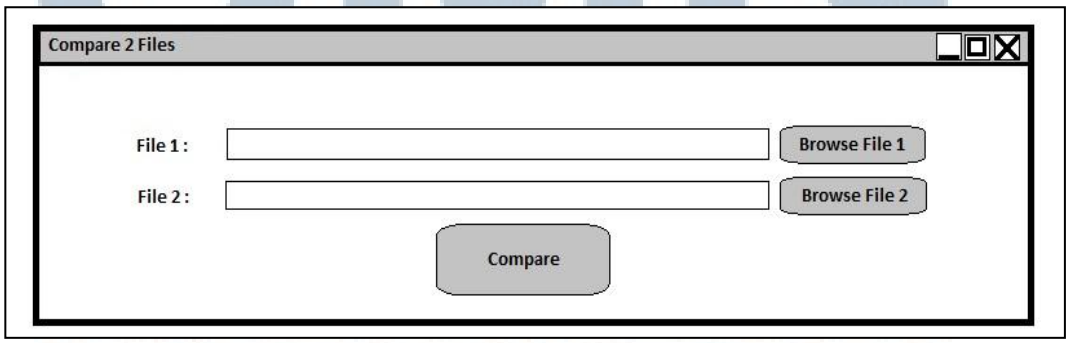
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.8 Desain Tampilan Antarmuka Halaman Utama Menu "*Help*"

2. Halaman "*2 Files*"

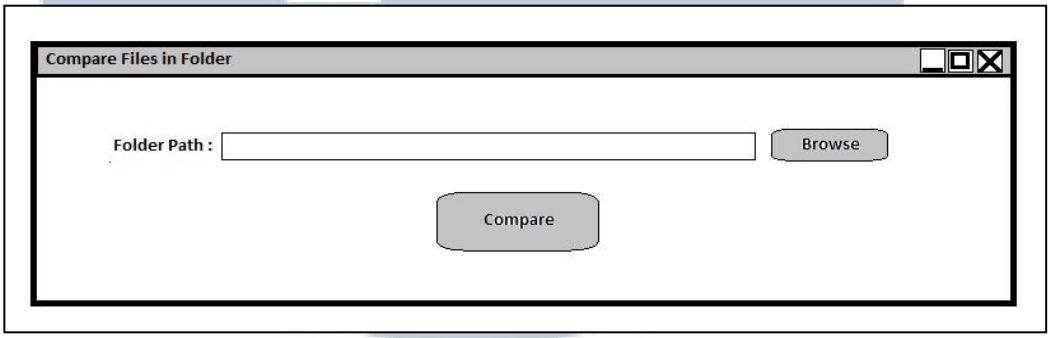
   Pada halaman "*2 Files*" terdapat tombol "*Browse File 1*" dan "*Browse File 2*" yang berfungsi untuk memilih *file* yang akan dibandingkan. Lalu terdapat pula tombol "*Compare*" yang berfungsi untuk memulai proses pembandingan antara 2 file tersebut. Hasil persentase pendeteksi dirancang akan muncul dalam bentuk *Dialog Box.*



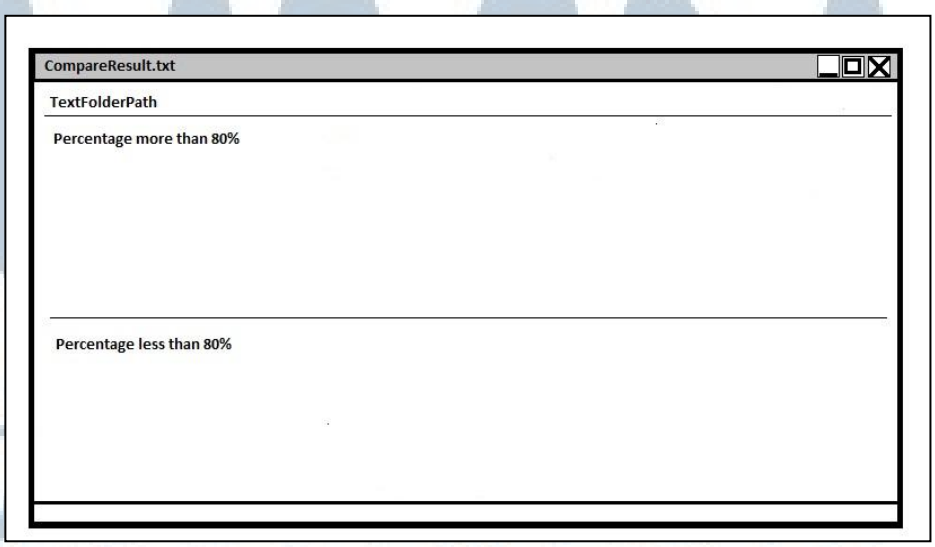
Gambar 3.9 Desain Tampilan Antarmuka Halaman "*Compare 2 Files*"

3. Halaman "*Files in Folder*"

   Pada halaman ini, terdapat tombol "*Browse*" yang berfungsi untuk memilih *folder* yang akan diproses. Selain itu, terdapat tombol "*Compare*" yang berfungsi untuk memulai proses pembandingan setiap *file* yang ada pada *folder* yang dipilih. Hasil persentase akan ditampilakan dalam satu *file* dengan format .txt yang diletakkan pada *folder* tempat *file* yang dibandingkan diletakkan.



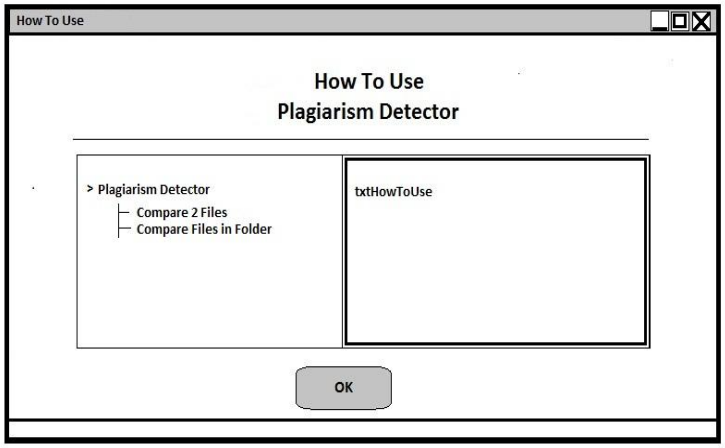
Gambar 3.10 Desain Tampilan Antarmuka Halaman "*Compare All Files in Folder*"



Gambar 3.11 Desain Tampilan Antarmuka Hasil dari Proses Pendeteksian dalam format .txt

4. Halaman "*How To Use*"

   Halaman "*How To Use*" terdiri dari sebuah *tree view* di bagian kiri dan penjelasan setiap *point* dalam *tree view* pada *text box* di bagian kanannya yang berfungsi untuk menampilkan informasi mengenai penggunaan aplikasi serta sebuah tombol "*OK*" yang berfungsi untuk menutup halaman ini.

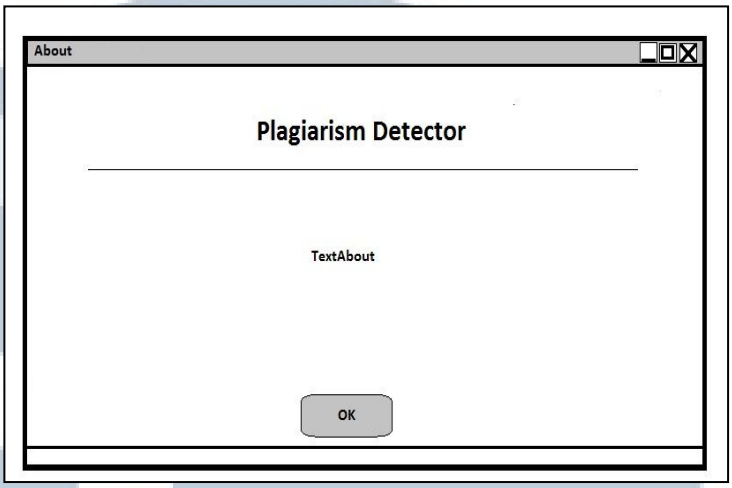


Gambar 3.12 Desain Tampilan Antarmuka Halaman "*How To Use*"

5. Halaman "*About*"

   Halaman "*About*" berisi informasi mengenai gambaran umum mengenai aplikasi dan profil singkat mengenai pembuat aplikasi dan terdapat sebuah tombol "OK" yang berfungsi untuk menutup halaman ini.

UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.11 Desain Tampilan Antarmuka Halaman "*About*"

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA