BAB III

METODE DAN PERANCANGAN SISTEM

3.1 Metode Penelitian

Metode penelitian yang digunakan dalam membangun dan mengimplementasikan algoritma *hamming distance* dalam aplikasi pendeteksi plagiarisme *source code* bahasa C adalah sebagai berikut.

1. Studi Literatur

Pada tahap ini, dilakukan pencarian dan pembelajaran berbagai literatur dan konsep-konsep yang berkaitan dengan rumusan masalah, teori-teori yang berhubungan dengan teknologi dan bahasa pemrograman yang sesuai, dan algoritma yang digunakan.

2. Pengumpulan Data

Pada tahap ini, dilakukan pengumpulan data-data dari artikel, jurnal ataupun karya ilmiah lainnya tentang algoritma yang dibahas.

3. Perancangan Aplikasi

Perancangan aplikasi dilakukan dengan pembuatan diagram yang dimaksudkan untuk memahami dan mendesain keseluruhan alur kerja aplikasi yang akan dibuat. Hal ini bertujuan untuk memperjelas seluruh detail aplikasi yang akan dibuat, sehingga pengerjaan aplikasi dapat berjalan dengan lancar dan sesuai rencana. Pada tahap ini juga dilakukan perancangan desain aplikasi.

4. Analisis Algoritma

Pada tahap ini akan dianalisis algoritma yang akan digunakan, yaitu algoritma *hamming distance*. Hal-hal yang akan dianalisis meliputi cara kerja

algoritma dan detail kegunaan algoritma, sehingga dapat diimplementasikan dengan optimal pada aplikasi ini.

5. Pembuatan Aplikasi

Pembuatan aplikasi dilakukan berdasarkan tujuan dan kegunaan aplikasi. Pembuatan program meliputi *interface, database,* dan koding aplikasi secara keseluruhan.

6. Testing dan Evaluasi

Pada tahap ini dilakukan *testing* pada aplikasi yang telah dibuat. *Testing* dilakukan dengan membandingkan kinerja algoritma dengan algoritma *levenshtein distance* pada contoh kasus serupa untuk mendapatkan kesimpulan mengenai kinerja algoritma *hamming distance*. Semua data uji akan dilampirkan pada laporan penelitian ini.

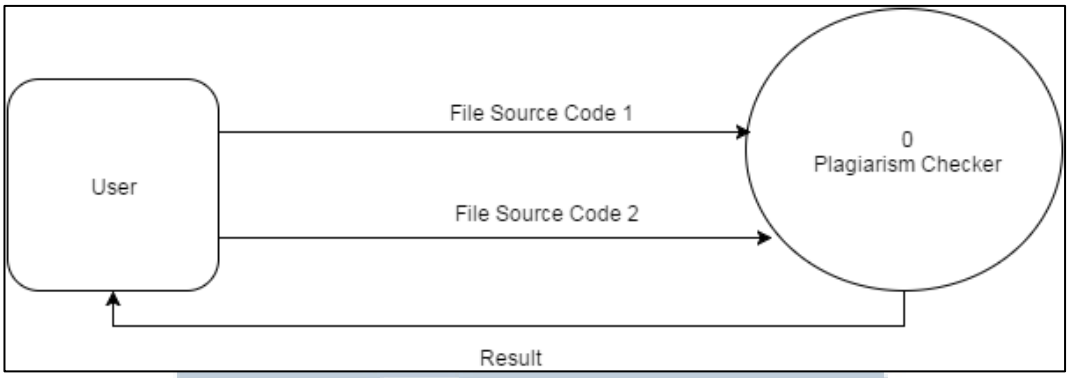
7. Penulisan Skripsi

Setelah semua langkah-langkah di atas telah selesai, maka langkah selanjutnya adalah penyusunan laporan skripsi sebagai dokumentasi.

3.2 Data Flow Diagram

3.2.1 Diagram Konteks

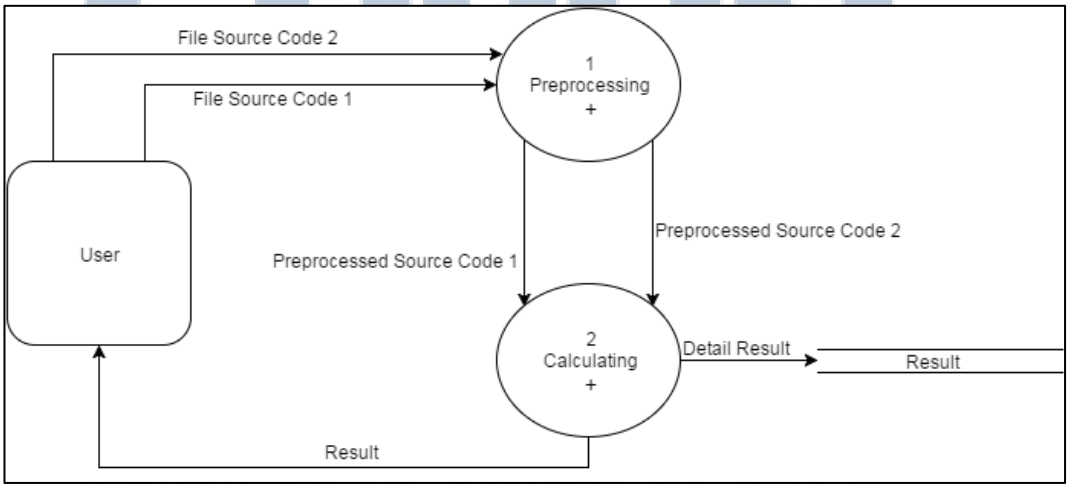
Aplikasi Plagiarism Checker memiliki satu buah entitas yang memberikan masukkan ke sistem berupa dua buah *file source code* bahasa C. Sedangkan sistem akan mengeluarkan *output* berupa persentase kesamaan antar kedua *file*. Gambar 3.1 menunjukkan diagram konteks aplikasi Plagiarism Checker.



Gambar 3.1 Diagram Konteks

3.2.2 Data Flow Diagram Level 1

Pada diagram *level* satu, terdapat dua buah proses utama yaitu proses "Preprocessing" dan proses "Calculate". Proses *preprocessing* akan menerima masukkan berupa *source code* bahasa C dan menghasilkan keluaran berupa *source code* yang telah ter-*preprocessing* untuk selanjutnya dihitung tingkat kemiripannya. Proses *calculate*, dapat menerima 2 masukkan, *source code* asli maupun *source code* yang telah melewati proses *preprocessing* sebelumnya. Gambar 3.2 menunjukkan *data flow diagram level* 1.

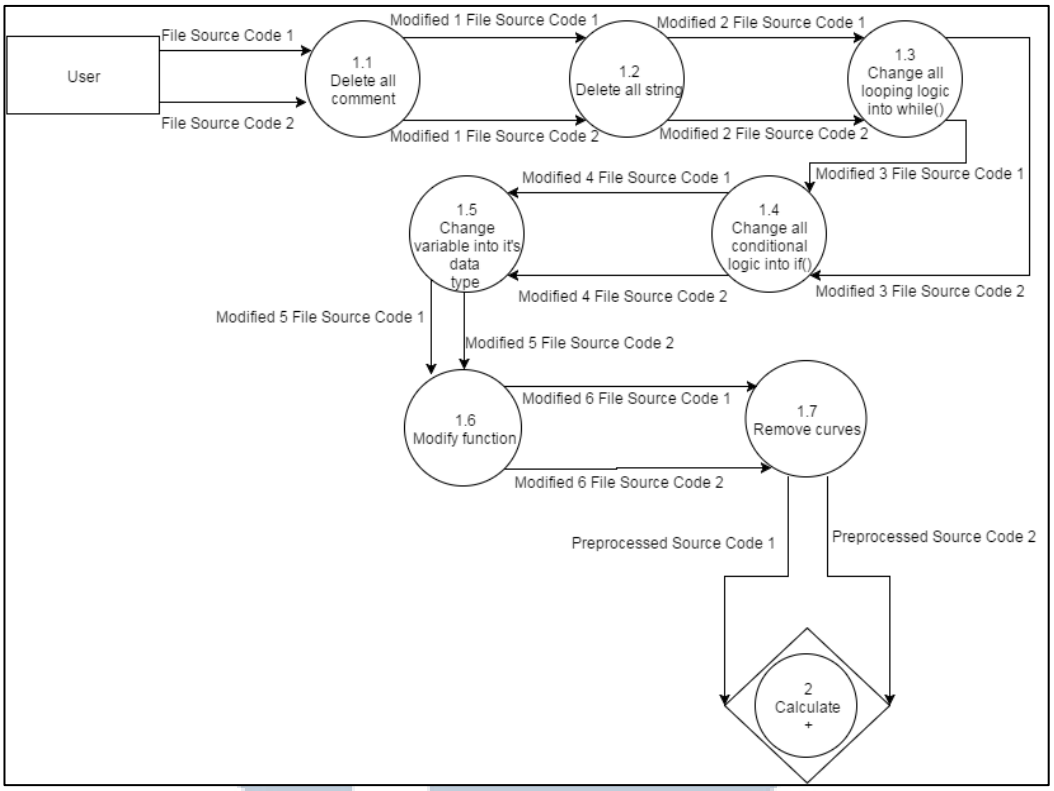


Gambar 3.2 *Data Flow Diagram Level* 1

3.2.3 Data Flow Diagram Level 2 Subproses Preprocessing

Pada proses *preprocessing*, terdapat 7 buah subproses. Subproses 1.1 berfungsi untuk menghapus semua komentar baik yang berada diantara tanda "/*....*/" maupun yang berada diantara tanda "//". Subproses 1.2 berfungsi untuk menghapus semua *string* yang berada diantara "....", yang biasa ditemui pada fungsi printf() dan scanf(). Subproses 1.3 berfungsi untuk mengubah logika *looping for* menjadi logika *looping while*. Hal ini penting untuk membantu mendeteksi plagiarisme dalam bentuk leksikal.

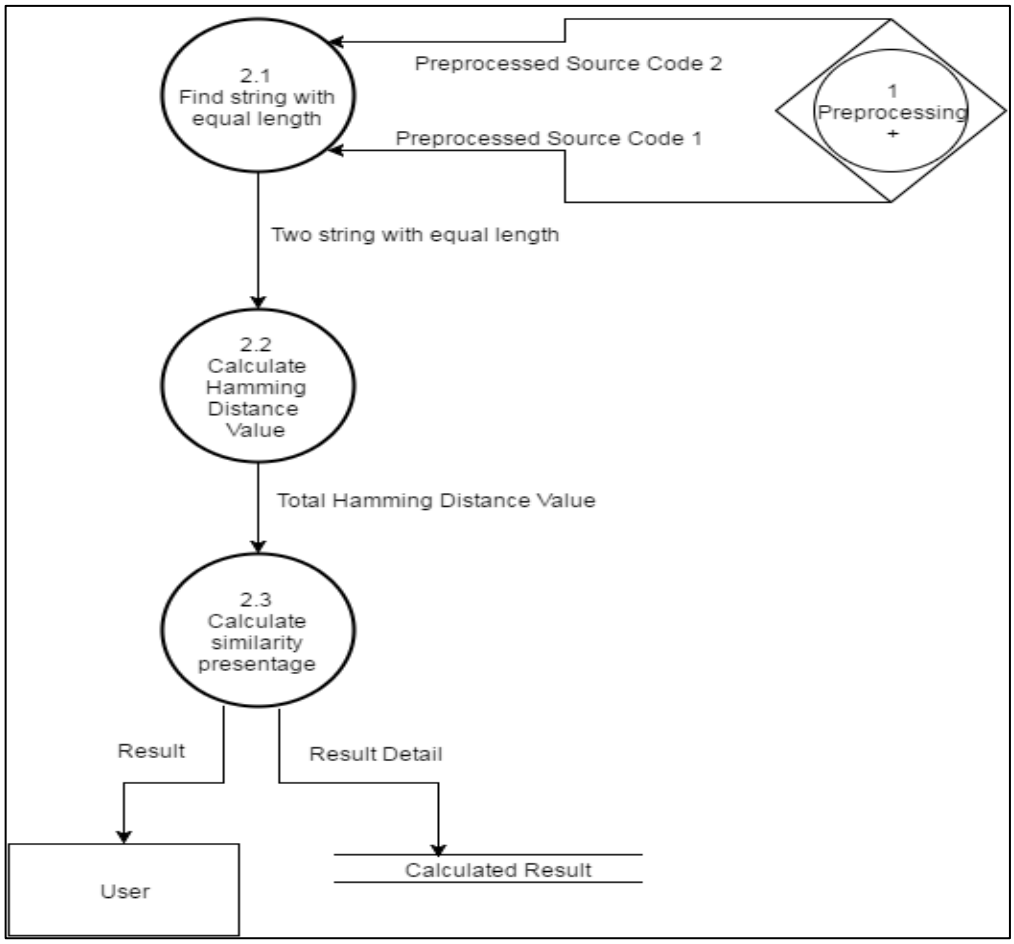
Subproses 1.4 berfungsi untuk mengubah logika kondisional *switch* menjadi logika kondisional *if*. Subproses 1.5 merupakan subproses yang berguna untuk mengubah penamaan variabel berdasarkan tipe data bentukannya. Hal ini diperlukan untuk membantu aplikasi mendeteksi plagiarisme dalam bentuk leksikal. Hasil dari subproses 1.5 adalah berupa *source code* yang isinya telah termanipulasi dan selanjutnya akan diserahkan ke proses *calculate*. Gambar 3.3 menunjukkan *data flow diagram level* 2 dari subproses *preprocessing*.



Gambar 3.3 *Data Flow Diagram Level* 2 Subproses *Preprocessing*

3.2.4 Data Flow Diagram Level 2 Subproses Calculate

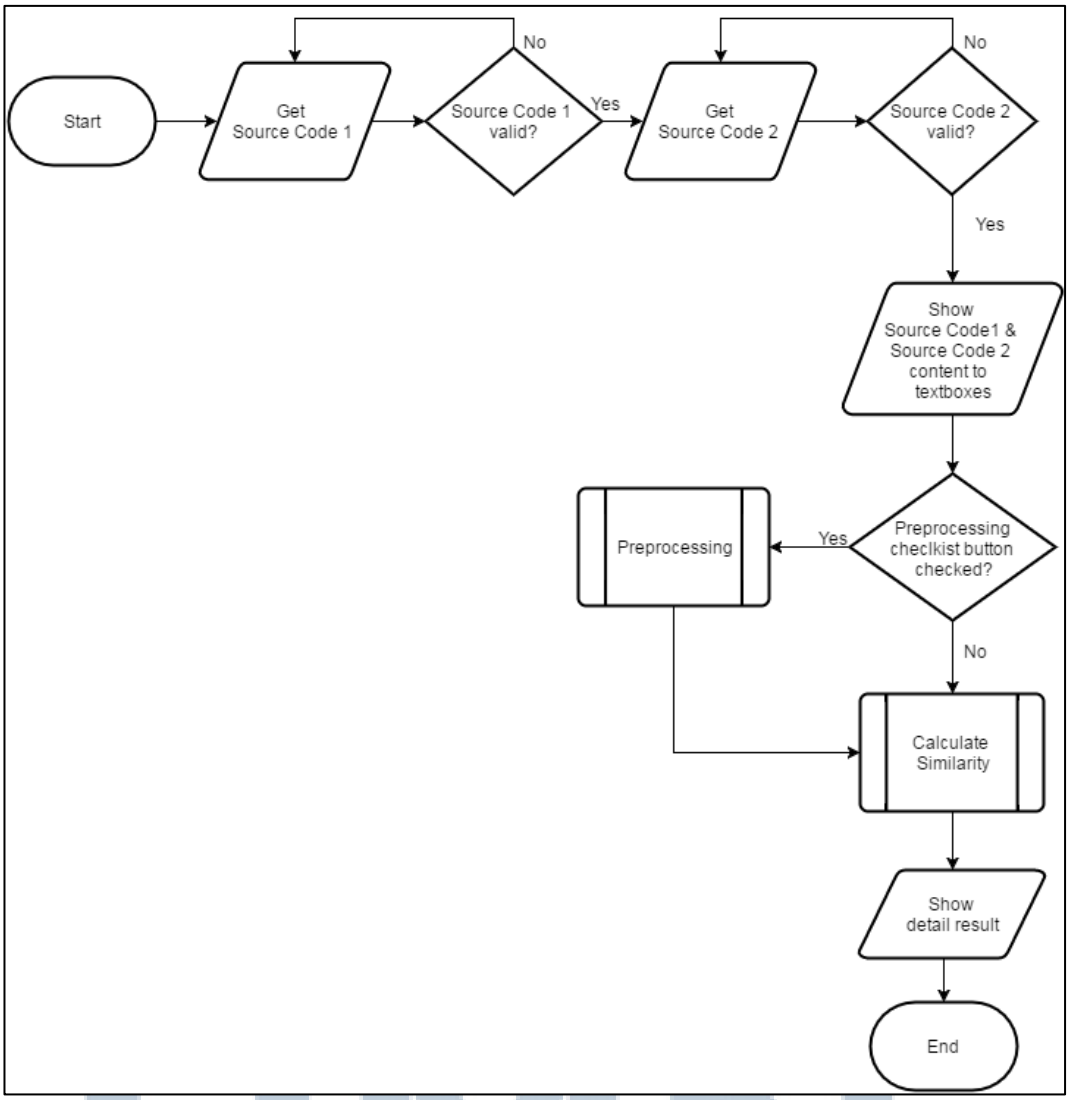
Pada proses *calculate*, terdapat tiga subproses. Subproses 2.1 berfungsi untuk mencari *string* dari kedua *file* yang memiliki panjang yang sama, hal ini dilakukan dengan menggunakan algoritma *brute force*. Subproses 2.2 berfungsi untuk menghitung nilai *hamming distance* antara kedua *string*, jika nilai *hamming distance* sama dengan 0, maka dipastikan bahwa kedua *string* identik. Jika kedua *string* identik, maka baris ke-n pada *file* ke-2 akan diubah menjadi *null* dan akan menambah total *hamming distance value*. Subproses 2.3 akan mengubah data yang diterima, ke dalam format persentase. Hasil keluaran dapat diberikan ke *user* sebagai keluaran sistem, sekaligus disimpan ke dalam *file* ".txt" beserta detailnya. Gambar 3.4 menunjukkan *data flow diagram level* 2 dari subproses *calculate*.



Gambar 3.4 *Data Flow Diagram Level* 2 Subproses *Calculate*

3.3 Diagram Alir

Pada saat aplikasi dibuka, program akan memilih 2 buah *source code* yang akan dibandingkan lalu memvalidasinya. Isi dari kedua *source code* tadi akan ditampilkan dalam 2 buah *textbox.* Setelah itu program akan memeriksa status *preprocessing*, yang apabila diaktifkan, maka program akan mengirim seluruh isis *source code* ke dalam prosedur *preprocessing*. Selanjutnya program memulai proses perhitungan. Setelah aplikasi selesai menghitung, aplikasi akan menampilkan hasil perhitungan dalam bentuk persentase jumlah banyaknya baris yang memiliki kemiripan. Gambar 3.5 menunjukkan diagram alir dari aplikasi Plagiarism Checker.

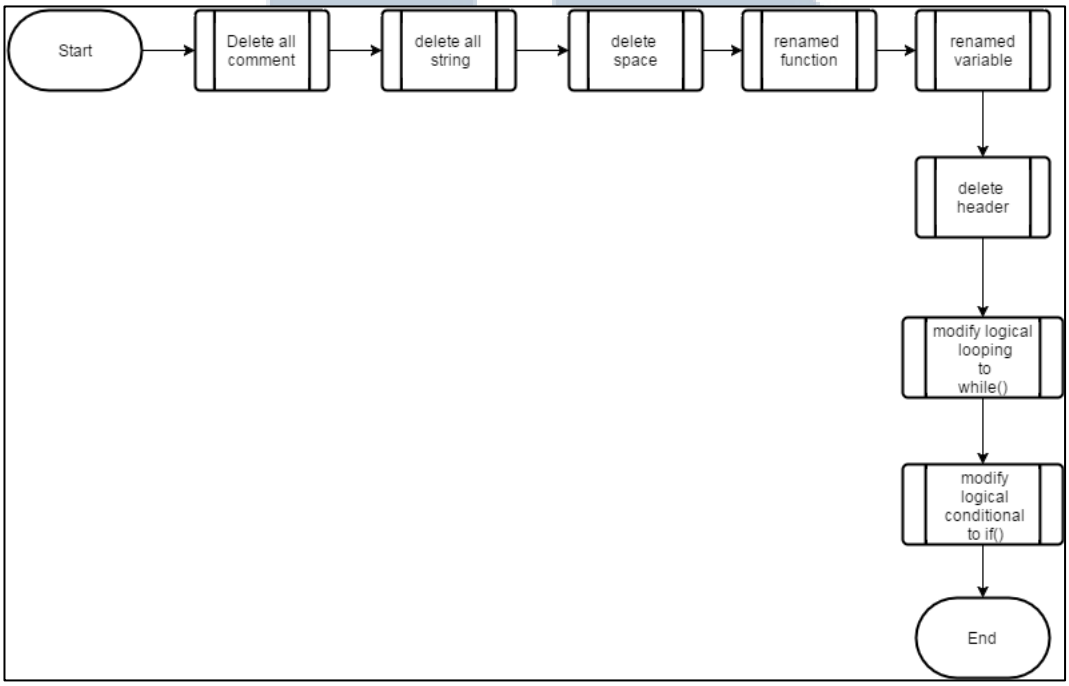


Gambar 3.5 Diagram Alir Program Plagiarism Checker

3.3.1 Preprocessing

Jika *user* memilih untuk menggunakan fitur *preprocessing*, maka program akan memodifikasi isi dari *source code* yang bersangkutan. Namun modifikasi ini tidak akan disimpan secara permanen di dalam *source code* aslinya, sehingga teknik ini tidak akan mengubah isi *source code* aslinya. Beberapa modifikasi yang dilakukan antara lain menghapus komentar, menghapus *string*, menghapus spasi, mengubah penamaan fungsi, mengubah penamaan variabel, mengubah bentuk

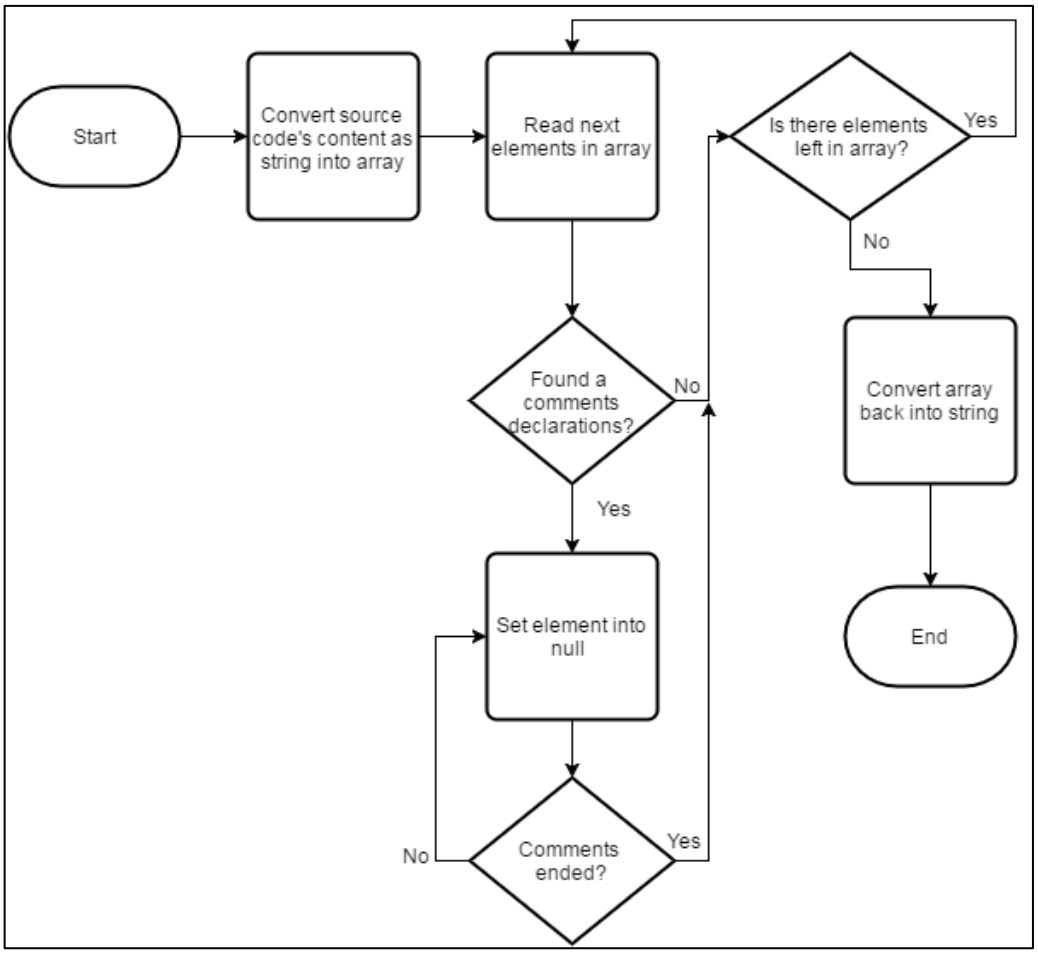
logika pengulangan menjadi fungsi *while(),* mengubah bentuk logika kondisional menjadi fungsi *if().* Gambar 3.6 menunjukkan diagram alir dari proses *preprocessing*.



Gambar 3.6 Diagram Alir *Preprocessing Source Code*

A. Delete All Comment

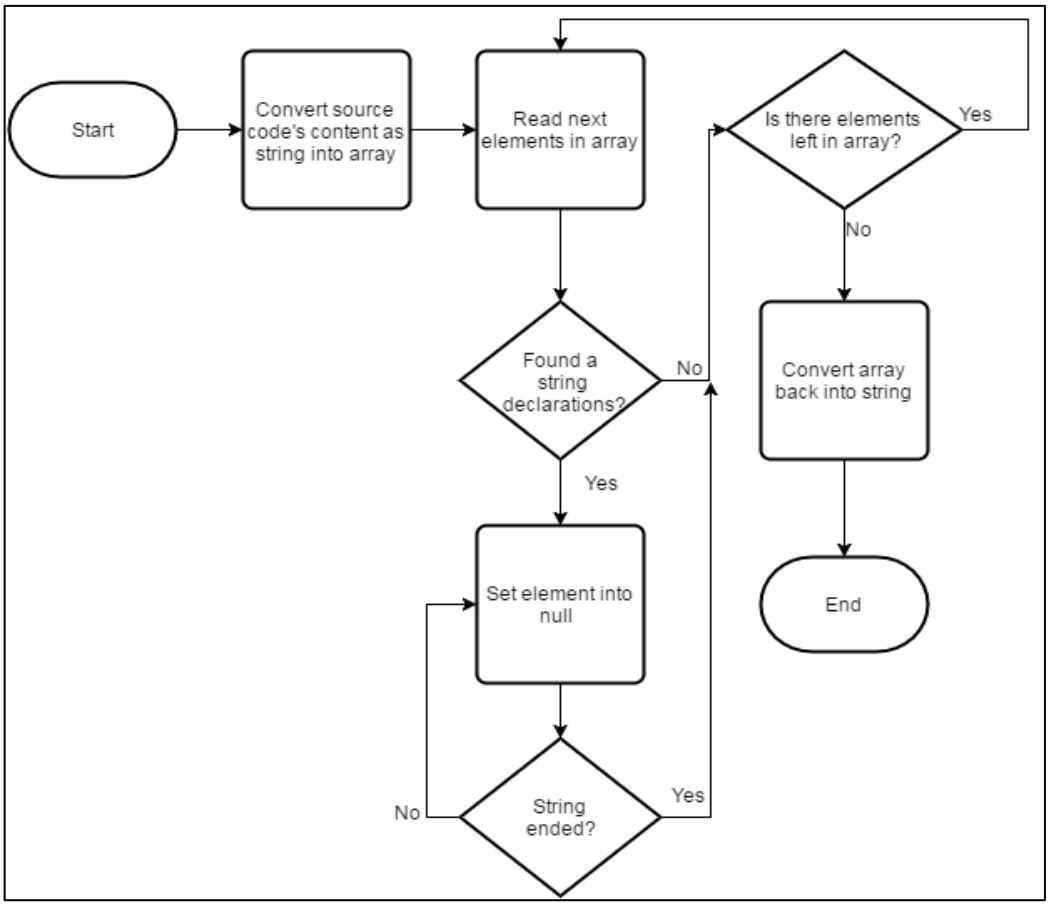
Pada proses ini, seluruh komentar baik yang bersifat *inline* maupun *multi line* pada *source code* akan dihapus. Hal pertama yang dilakukan program adalah mengubah kontent *source code* dari bentukkan *string* menjadi array, hal ini untuk mempermudah dalam mendeteksi deklarasi komentar, khususnya pada jenis *multiline.* Selanjutnya, setiap elemen komentar akan diubah menjadi *null* sampai menemukan akhir dari komentar (“*/”). Gambar 3.7 menunjukkan diagram alir proses *delete all comment*.



Gambar 3.7 Diagram Alir Proses *Delete All Comment*

B. Delete All *String*

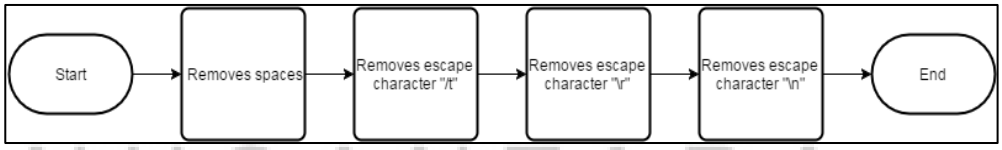
Pada proses ini, seluruh *string* yang berada diantara tanda petik dua ("...") akan dihilangkan, termasuk *string* yang berada di dalam fungsi *printf* dan *scanf*. Pada dasarnya, logika fungsi ini memiliki logika yang mirip dengan alur program Delete All Comment. Hanya saja, yang menjadi patokan dekalrasi string disini adalah tanda petik dua (") yang sekaligus menjadi akhir dari sebuah *string*. Gambar 3.8 menunjukkan diagram alir proses *delete all string*.



Gambar 3.8 Diagram Alir Proses *Delete All String*

C. Delete Space

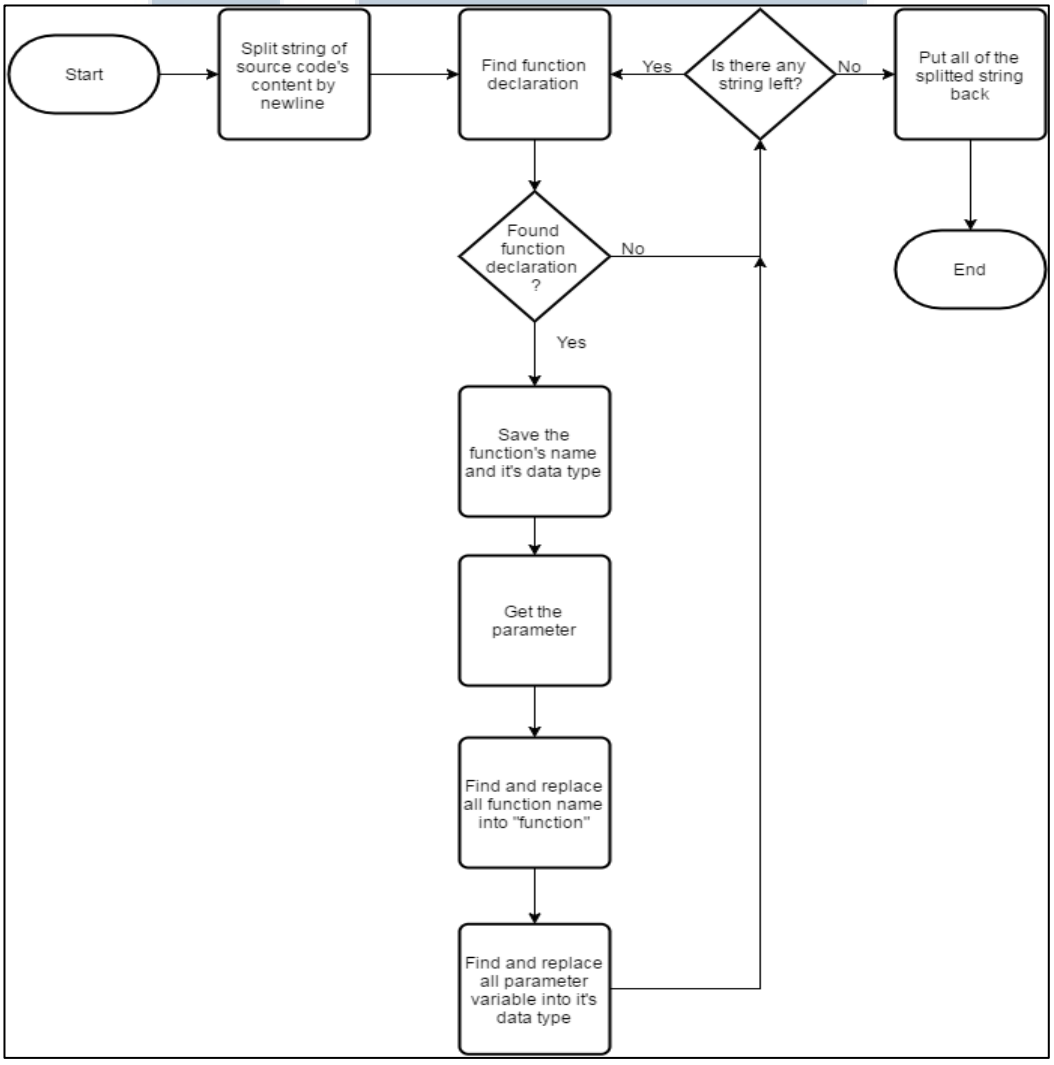
Pada proses ini, seluruh *blank space* akan dihilangkan. Jenis *blank space* yang dimaksud adalah *space* dan *escape character* “/r”, “/t” dan “/n”. Penghapusan ini sendiri menggunakan *library string* yang telah tersedia di Visual C# untuk memanipulasi *string*. Gambar 3.9 menunjukkan diagram alir proses *delete space*.



Gambar 3.9 Diagram Alir Proses *Delete Space*

D. Renamed Function

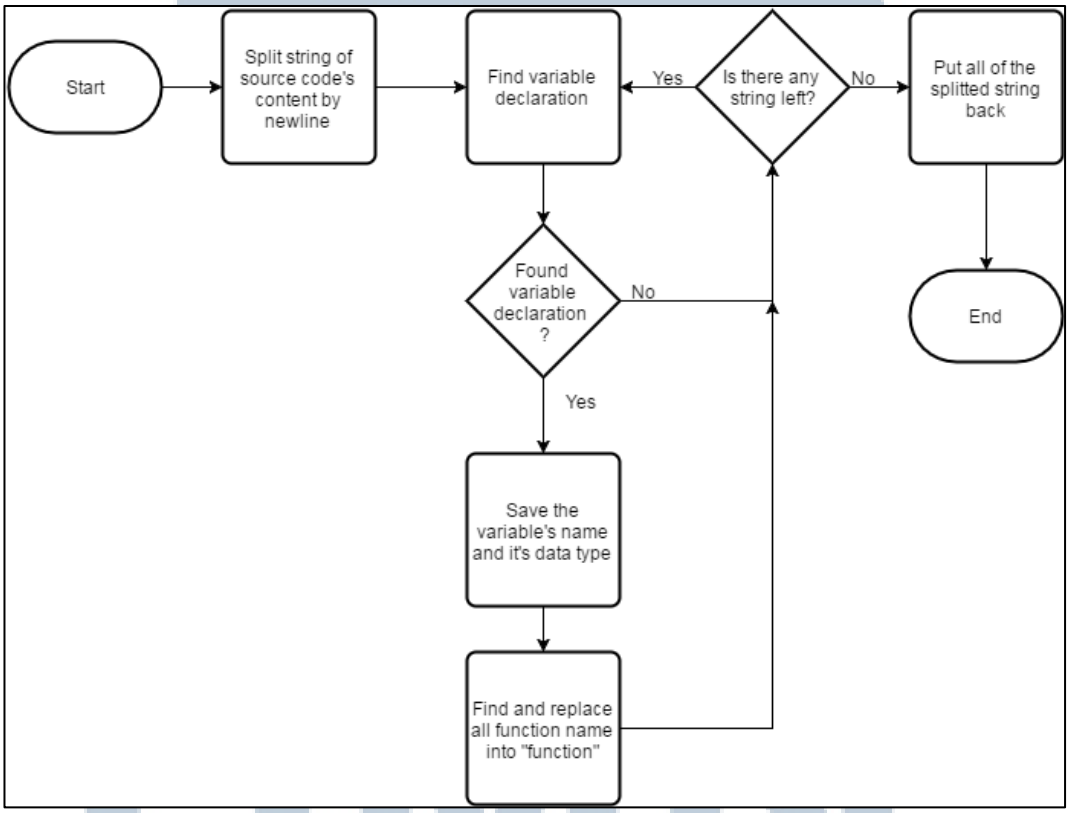
Pada proses ini dilakukan perubahan nama fungsi menurut tipe data nilai kembalinya (*return value*), termasuk mengubah nama parameternya menjadi tipe data bentukannya. Seluruh isi *source code* akan dipisahkan ("split") bedasarkan *newline*-nya, menggunakan manipulasi *string*. Jika ditemukan adanya *function*, maka nama fungsi, dan parameternya akan disimpan dalam sebuah variabel. Selanjutnya akan dicari pemanggilan nama fungsi atau parameter untuk diubah menurut tipe data bentukannya. Gambar 3.10 menunjukkan diagram alir proses *renamed function.*



Gambar 3.10 Diagram Alir Proses *Renamed Function*

E. Renamed Variable

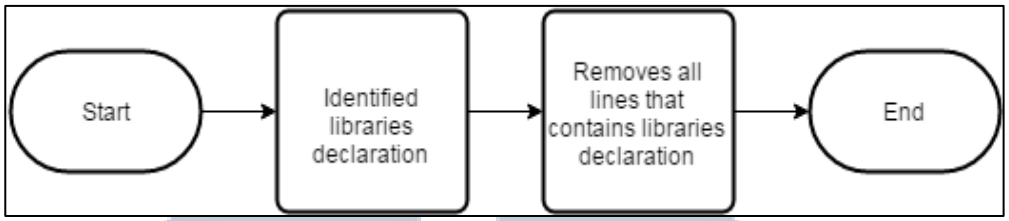
Proses ini memiliki konsep yang sama dengan proses *renamed function*, yaitu mengubah nama variabel menjadi tipe data bentukannya. Sayangnya, proses ini belum mendukung pengubahan tipe data *struct*. Gambar 3.11 menunjukkan diagram alir proses *renamed variable*.



Gambar 3.11 Diagram Alir Proses *Renamed Variable*

F. Delete Header

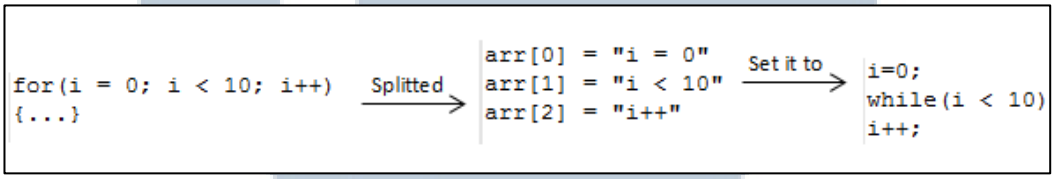
Pada proses ini akan dilakukan peghapusan baris kode yang mengadung deklarasi *libraries* yang digunakan. Gambar 3.12 menunjukkan diagram alir proses *delete header*.



Gambar 3.12 Diagram Alir Proses *Delete Header*

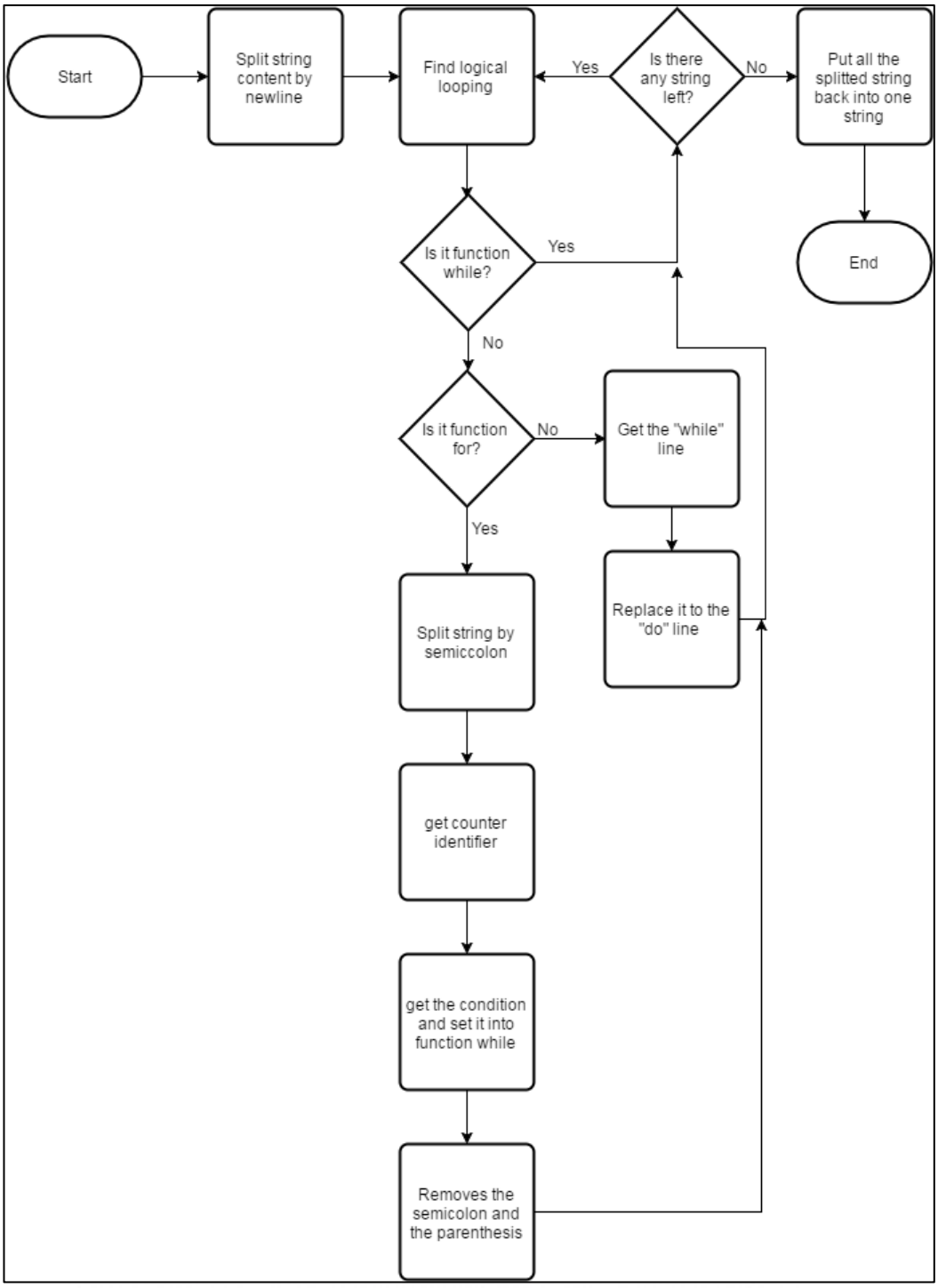
G. Modify Logical Looping to While

Pada proses ini, seluruh fungsi logika *looping* seperti fungsi *for* dan *do-while* akan diubah menjadi fungsi *while*. Gambar 3.13 menunjukkan cara kerja ubahan fungsi *for* menjadi fungsi *while.*



Gambar 3.13 Cara Kerja Ubahan Fungsi *For* Menjadi Fungsi *While*

Pada perubahan fungsi *for* menjadi fungsi *while*, terdapat beberapa langkah-langkah yang harus dilakukan. Langkah pertama adalah dengan memisahkan baris *for* bedasarkan semikolonnya, setelah itu didapat 3 data penting berupa identifikasi kounter, kondisional, dan penambahan kounter. Selanjutnya terapkan fungsi *while* dengan menggunakan data yang didapat diatas. Gambar 3.14 menunjukkan diagram alir proses *modify logical looping to while*.

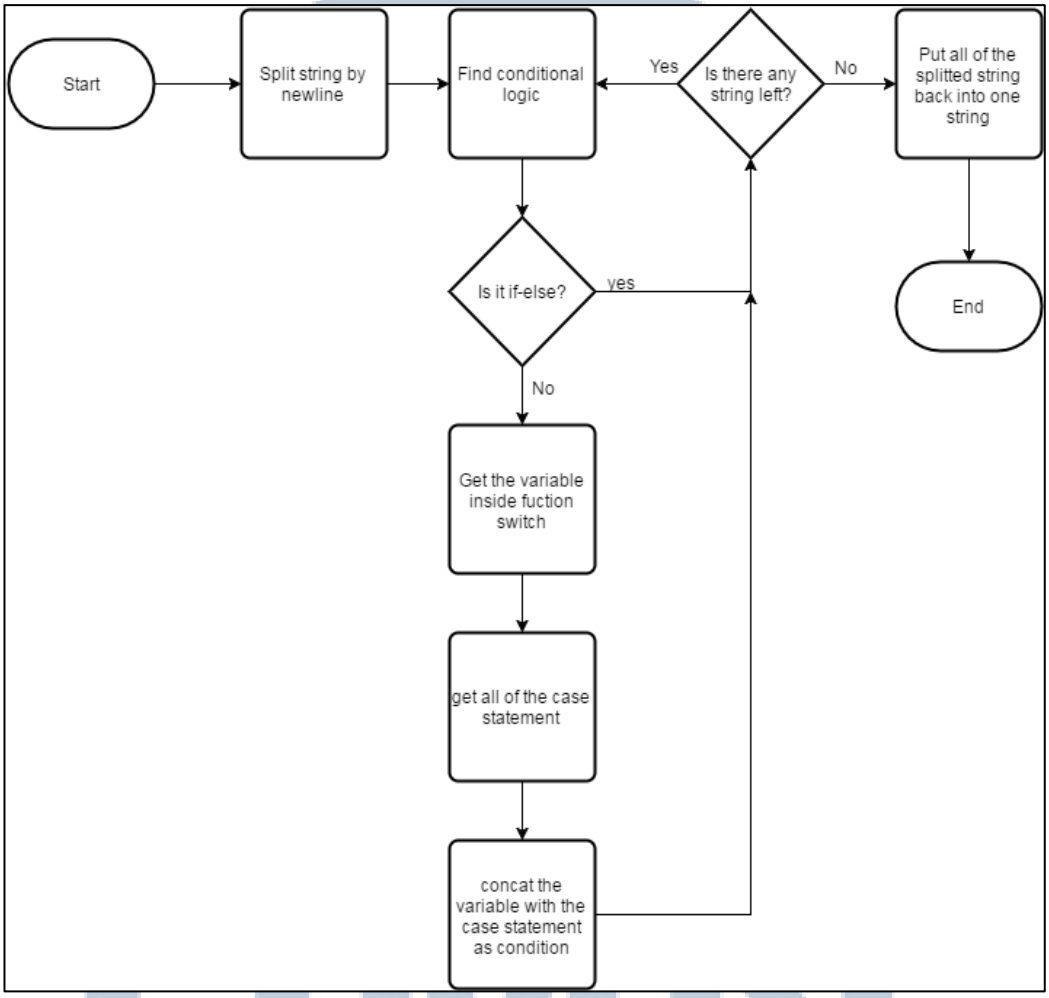


Gambar 3.14 Diagram Alir Proses *Modify Logical Looping To While*

H. Modify Logical Conditional to If

Proses ini memiliki cara kerja yang sama seperti proses pada *point* I, hanya saja proses ini diimplementasikan pada *conditional logic*. Beberapa *conditional*

logic antara lain *switch-case* dan *if*. Gambar 3.15 menunjukkan diagram alir proses *modify logical conditional to if*.

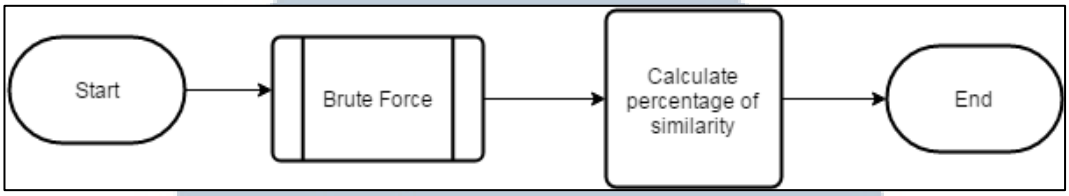


Gambar 3.15 Diagram Alir Proses *Modify Logical Conditional To If*

3.3.2 Calculate Similarity Persentage

Selanjutnya jika *file* sudah siap dibandingkan, maka sistem akan mulai menghitung persentase tingkat kemiripan kedua *file*. Pada proses ini, prosedur *brute force* dipanggil, yang nantinya akan menghasilkan jumlah baris yang mirip antar dua buah *source code*. Selanjutnya, jumlah baris kode yang mirip tadi akan dipersentasekan terhadap jumlah baris kode terbanyak dari antara kedua *source*

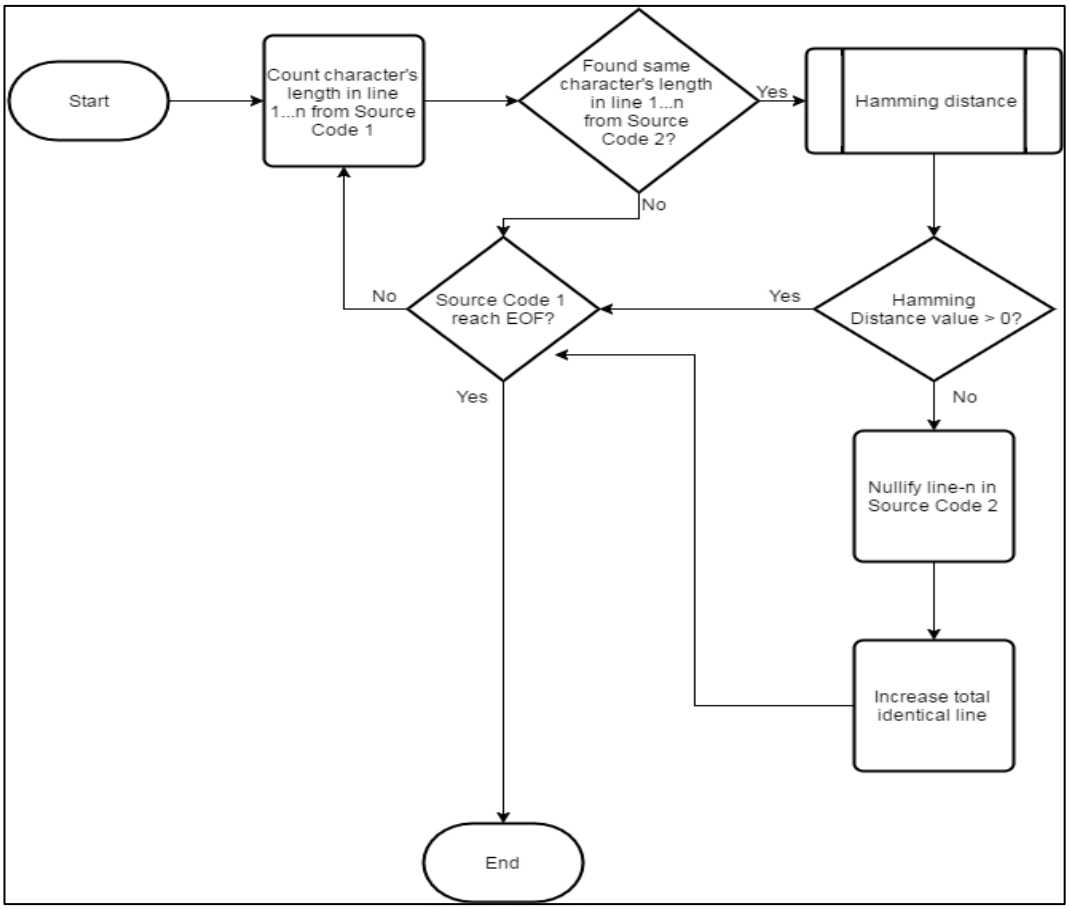
code input. Gambar 3.16 menunjukkan diagram alir *calculate similarity persentage*



Gambar 3.16 Diagram Alir *Calculate Similarity Persentage*

A. Brute Force

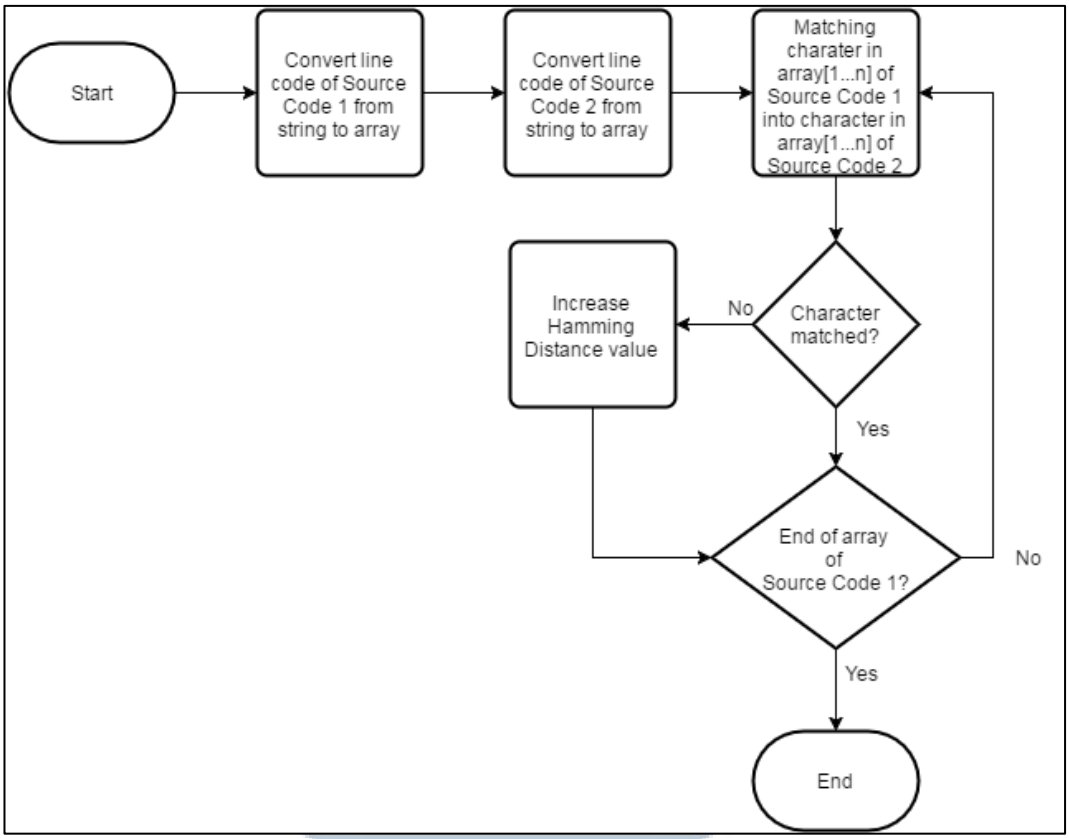
Pada subproses ini, algoritma *brute force* digunakan untuk memilah baris kode pada *source code* pertama yang memiliki kecocokkan panjang karakter dengan baris kode pada *source code* kedua. Jika sepasang baris kode telah ditemukan, maka akan dilanjutkan ke proses *hamming distance*. Dari proses *hamming distance*, jika nilai yang didapat sama dengan nol, maka artinya kedua baris kode tersebut terdeteksi mirip satu sama lain. Selanjutnya program akan menandai baris kode pada *source code* 2, agar tidak lagi dicocokkan dengan baris kode pada *source code* 1. Penandaan tersebut dilakukan dengan mengubah baris kode pada *source code* 2 menjadi *null*. Setelah itu jumlah baris yang mirip akan ditambah dan ditampung sementara untuk dicari nilai totalnya. Proses *brute force* ini akan diulang sebanyak baris kode pada *source code* 1. Gambar 3.17 menunjukkan diagram alir dari proses algoritma *brute force*.



Gambar 3.17 Diagram Alir Algoritma Brute Force

B. Hamming Distance

Pada proses ini algoritma *hamming distance* telah siap mendeteksi dua buah *string* dengan panjang yang sama. Hal yang pertama dilakukan adalah memecah *string* yang didapat menjadi bentukkan *array* sehingga akan mempermudah pendeteksian kata per kata. Setelah itu program memeriksa tiap elemen yang terdapat pada *array* dengan pencocokkan elemen-1 *array* 1 dengan elemen-1 *array* 2 sampai elemen-n *array* 1 dengan elemen-n *array* 2. Setiap elemen *array* yang terdeteksi berbeda akan menambah nilai *hamming distance*-nya. Gambar 3.18 menunjukkan diagram alir algoritma *hamming distance*.



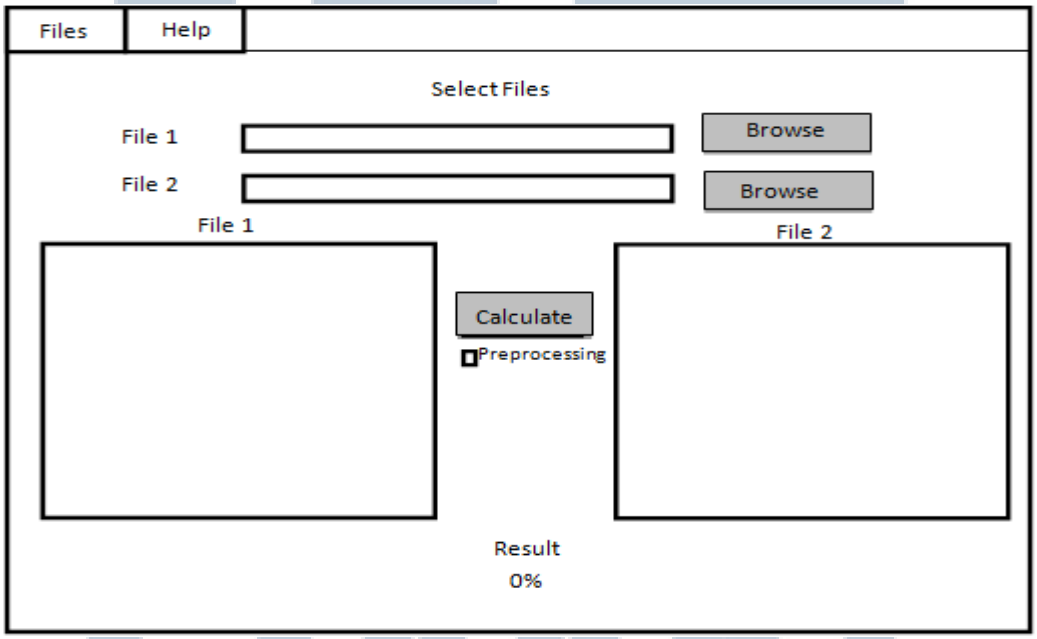
Gambar 3.18 Diagram Alir Algoritma Hamming Distance

3.4 Rancangan Antarmuka

3.4.1 Tampilan Awal

Saat pertama kali membuka aplikasi, *user* akan langsung diminta memasukan dua buah *file* yang akan dibandingkan. Di atas jendela aplikasi, terdapat *menu strip* pada pojok kanan atas tampilan yang menampilkan menu *file* dan menu *help*. Pada menu *file*, terdapat pilihan untuk menyimpan hasil perhitungan ke dalam format '.txt' sedangkan pada menu *help, user* dapat melihat halaman About dan menu *how to use*. Selanjutnya terdapat tombol "Browse" yang berfungsi untuk memilih *source code* yang akan dihitung tingkat plagiarismenya. Di tengah jendela aplikasi terdapat dua buah *textbox* yang akan menampilkan isi dari masing-masing *source code*. Sedangkan di antara kedua *textbox* tadi, terdapat

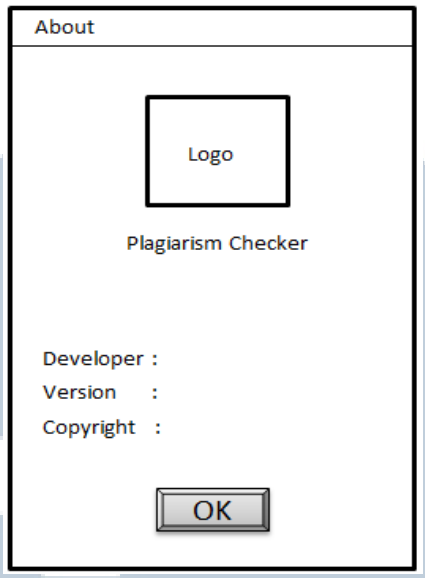
checklist button yang akan memberikan *user* kesempatan untuk mem-*preprocessing source code* sebelum dihitung. Di atas *checklist preprocessing*, terdapat tombol "Calculate" yang berfungsi untuk memulai perhitungan. Selanjutnya, hasil perhitungan dapat dilihat di bawah jendela pada *section result*. Secara *default, result* akan menampilkan 0%. Gambar 3.19 menunjukkan tampilan dari halaman awal.



Gambar 3.19 Rancangan Tampilan Awal

3.4.2 Tampilan About

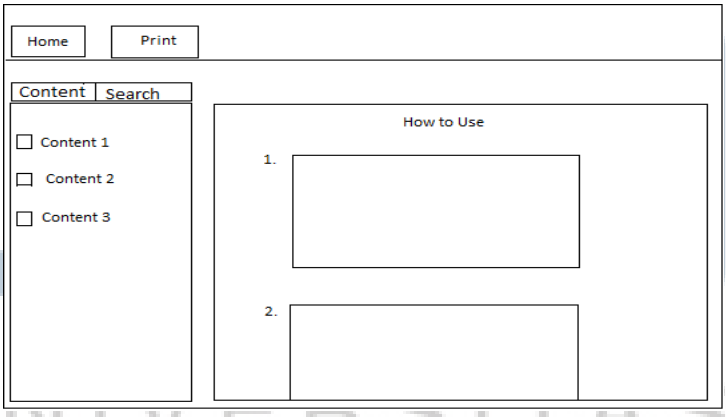
Pada halaman About, *user* dapat melihat deskripsi aplikasi, berupa nama pembuat, versi aplikasi, dan tahun pembuatan. Gambar 3.20 menunjukkan tampilan halaman About.



Gambar 3.20 Rancangan Tampilan About

3.4.2 Tampilan How To Use

Pada halaman ini, *user* dapat mencari jawaban atas beberapa pertanyaan mengenai aplikasi serta mengetahui *step by step* cara menggunakan aplikasi. Gambar 3.21 menunjukkan rancangan halaman *how to use*.



Gambar 3.21 Rancangan Halaman *How To Use*