BAB IV

PERANCANGAN

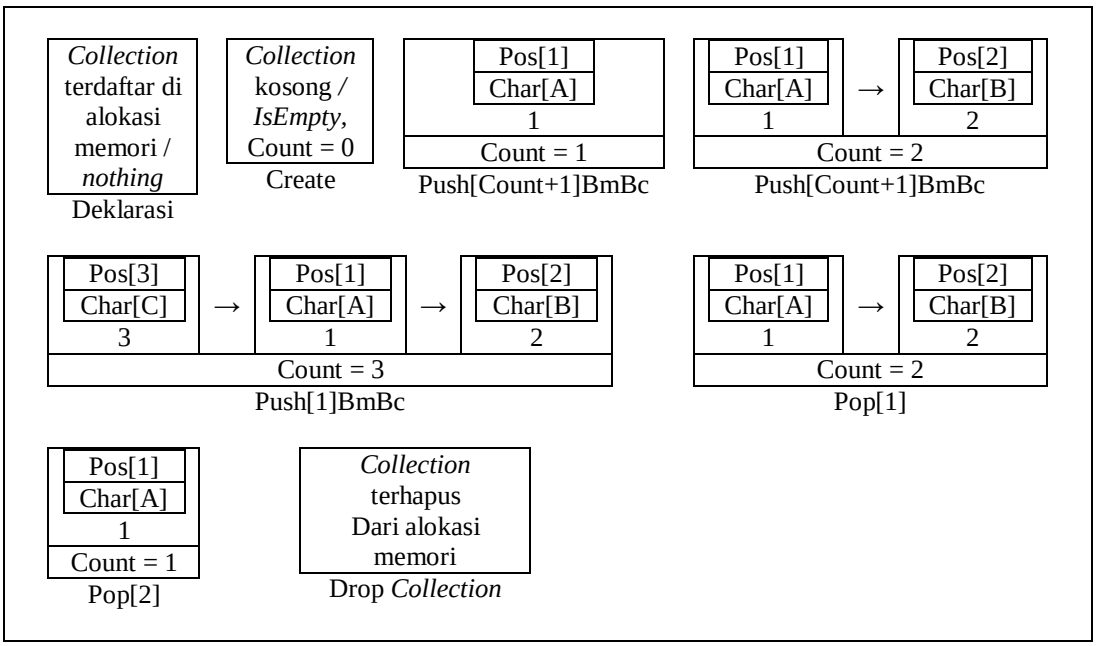
4.1. Implementasi *Stack* dengan *Single Linked List* Sebagai *Collection Control* Pada Translator

Pengggunaan *stack* dengan *single link list* dimaksudkan sebagai manajemen memori pada *runtime* Translator. *Single link list* difungsikan sebagai media penyimpanan data sementara hasil proses algoritma Boyer-Moore. Dimana *stack* memiliki nilai batas penyimpanan yang dinamis. Sehingga dapat memaksimalkan alokasi memori terhadap kebutuhan Translator yang sedang berjalan. Di Visual Basic 6.0, *stack* dengan *single link list* dapat diimplementasikan sebagai *collection control*. Operasi-operasi yang diterapkan pada collection antara lain :

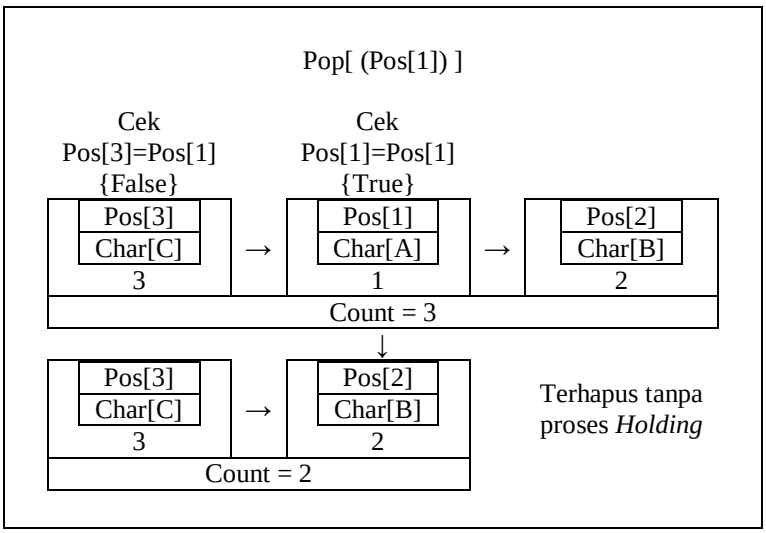
1. Deklarasi yaitu proses pendaftaran *collection control* pada alokasi memori.
2. Push, yaitu operasi menambahkan elemen pada urutan terakhir, di awal ataupun di akhir tumpukan dengan *class module* sebagai implementasi *record* pada elemen.
3. Pop, yaitu operasi mengambil sebuah elemen data pada urutan terujung dan menghapus elemen tersebut dari *collection*.
4. IsEmpty, yaitu proses pemeriksaan apakah *collection* dalam keadaan kosong.
5. Create, yaitu proses pembuatan *collection* kosong, dengan pemberian nilai *as new collection*.

6. Drop, yaitu operasi penghapusan *collection* dari alokasi memori, dengan pemberian nilai *nothing*.

Berikut representasi metode pemanfaatan *collection* untuk mendukung kinerja Translator dengan diasumsikan BmBc sebagai *class module*, Pos dan Char sebagai elemen *class module.*



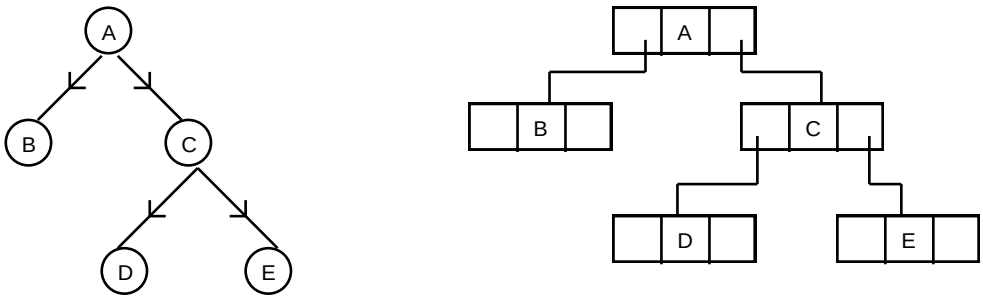
Gambar 4.1 Operasi standar pada *collection* Visual Basic 6.0.



Gambar 4.2 Kelebihan operasi *collection Control* pada Visual Basic 6.0.

4.2. Implementasi *Binary Tree* Sebagai *Class Modules Control* Pada Translator

Sama halnya dengan implementasi *stack, binary tree* difungsikan sebagai solusi pemecahan hirarki operator pada sebuah ekspresi.



Gambar 4.3 Representasi *binary tree*

Pada implementasinya, *binary tree* hanya digunakan sebagai tempat penyimpanan sementara dari potongan-potongan teks yang dikenali sebagai sebuah ekpresi. Berikut algoritma penerapan *binary tree* pada translator:

1. Lakukan pencarian posisi dengan menggunakan prosedur BM untuk 2 kata kunci pada teks. Kata kunci-1 merupakan kata kunci dengan arti buka (tanda yang menyebutkan sebuah awalan) dan Kata kunci-2 merupakan kata kunci dengan arti tutup (tanda yang menyebutkan sebuah akhiran). Contoh, kata kunci-1 : "(" dan kata kunci-2 : ")"
2. Jika jumlah posisi kedua kata kunci yang ditemukan tidak sama (tidak sepasang) maka berikan pesan kesalahan.
3. Jika jumlah posisi kedua karakter sama, maka bandingkan antara posisi paling pertama dari kata kunci-1 terhadap posisi paling pertama dari kata kunci-2 pada teks.

4. Jika kata kunci-1 memiliki nilai perbandingan terbesar, maka proses ulang (*rekursif*) teks dengan menambahkan kata kunci awalan dan akhiran pada kedua ujung teks.
5. Tambahkan setiap nilai posisi yang ditemukan ke dalam *binary tree*. *Binary tree* pada posisi ini merupakan *binary tree* yang tersusun dengan 1 *subtree* kanan, 0 *subtree* kiri. Difungsikan untuk memudahkan dalam pengurutan *node*.
6. Urutkan setiap posisi yang berkesesuaian (pasangan) antara posisi kata kunci-1 dan 2 dengan menggunakan metode pengurutan *Pre-Order*. *Pre-Order* dimaksudkan agar pengolahan operasi terjadi dari posisi buka-tutup terluar (hirarki terkecil). Langkah *Pre-Order* dilakukan dengan mengunjungi root *Bineary Tree*, mengunjungi *subtree* kiri, lalu *subtree* kanan secara rekursif.
7. Pada setiap proses pengunjungan, dilakukan perubahan bentuk pada *binary tree* hingga menjadi sebuah *binary tree transversal* dengan *Pre-Order* sebagai tolok ukur struktur *tree*.
8. Setelah posisi-posisi kata kunci-1 dan 2 telah tersusun sebagaimana semestinya, terjemahkan bagian teks yang berada pada posisi yang berkesesuaian terhadap masing-masing posisi pada *binary tree.* Proses pengunjungan *subtree* dilakukan dengan metode *Post-Order* secara rekursif, dengan tujuan agar teks dapat terpecah dari bagian paling dalam (hirarki terbesar).

4.3. Implementasi Algoritma Boyer-Moore pada Translator

Fungsi utama pada Translator terhadap teks yang dibentuk menjadi pernyataan atau serangkaian pernyataan Pascal adalah pencocokan suatu *pattern* tertentu yang diasumsikan sebagai pemisah rangkaian pembentuk pernyataan Pascal. Pencocokan ini dilakukan untuk memenggal teks berdasarkan posisi *pattern* terhadap teks, sehingga teks dapat dikenali sebagai pernyataan Pascal dan diterjemahkan ke pernyataan C.

Kaidah pertama dari algoritma Boyer-Moore ini adalah masing-masing karakter dari *pattern* pencocokan harus mempunyai kode ASCII yang sama terhadap target karakter yang ada pada teks serta karakter-karakter antara *pattern* dan target harus terurut sama persis.

Kaidah yang kedua adalah hasil penerjemahan teks berdasarkan pencocokan *pattern* akan bernilai benar secara utuh jika teks telah sukses melalui proses compile pada compiler Turbo Pascal 7.0. Walaupun demikian, Translator menerapkan fungsi-fungsi yang dapat memastikan kebenaran sintaksis pernyataan berdasarkan perolehan posisi *pattern* terhadap teks dan mengabaikan beberapa kesalahan sederhana pada sintaksis sehingga dapat diterjemahkan ke bentuk C. Dengan kata lain, suatu teks memungkinkan diterjemahkan secara utuh, sebagian, atau tidak sama sekali. Untuk itu, dibutuhkan suatu basis pengetahuan dalam pencocokan posisi dan pemenggelan *pattern* terhadap teks.

4.3.1. Pengembangan Cara Kerja Umum Algoritma Boyer-Moore

Secara Umum, langkah-langkah dari teknik penerapan Boyer-Moore pada pencocokan *pattern* pada teks dapat direpresentasikan sebagai berikut :

1. Menjalankan prosedur preBmBc dan preBmGs untuk mendapatkan nilai perbandingan pergeseran.
    a. Menjalankan prosedur preBmBc. Fungsi dari prosedur ini adalah untuk menentukan berapa besar pergeseran yang dibutuhkan untuk mencapai karakter tertentu pada *pattern* dari karakter *pattern* terakhir/terkanan. Hasil dari prosedur preBmBc disimpan pada stack bmBc.
    b. Menjalankan prosedur preBmGs. Sebelum menjalankan isi prosedur ini, prosedur *suffix* dijalankan terlebih dulu pada *pattern*. Fungsi dari prosedur *suffix* adalah memenggal sejumlah karakter yang dimulai dari karakter terakhir/terkanan dengan sejumlah karakter yang dimulai dari setiap karakter yang lebih kiri. Hasil dari prosedur *suffix* disimpan pada stack suff. Dengan demikian suff[i] mencatat panjang dari *suffix* yang cocok dengan segmen dari *pattern* yang di akhiri karakter ke-i.
    c. Dengan prosedur preBmGs, dapat diketahui berapa banyak langkah pada *pattern* dari sebuah segmen ke segmen lain yang sama, dimana letaknya lebih kiri dengan karakter di sebelah kiri segmen yang berbeda. Prosedur preBmGs menggunakan stack *suff* untuk mengetahui semua pasangan segmen yang sama.
2. Menjalankan prosedur BM, proses pencocokan *pattern* dengan menggunakan hasil dari prosedur preBmBc dan preBmGs yaitu stack bmBc dan bmGs sebagai landasan keputusan pergeseran.

4.3.1.1. Prosedur preBmBc

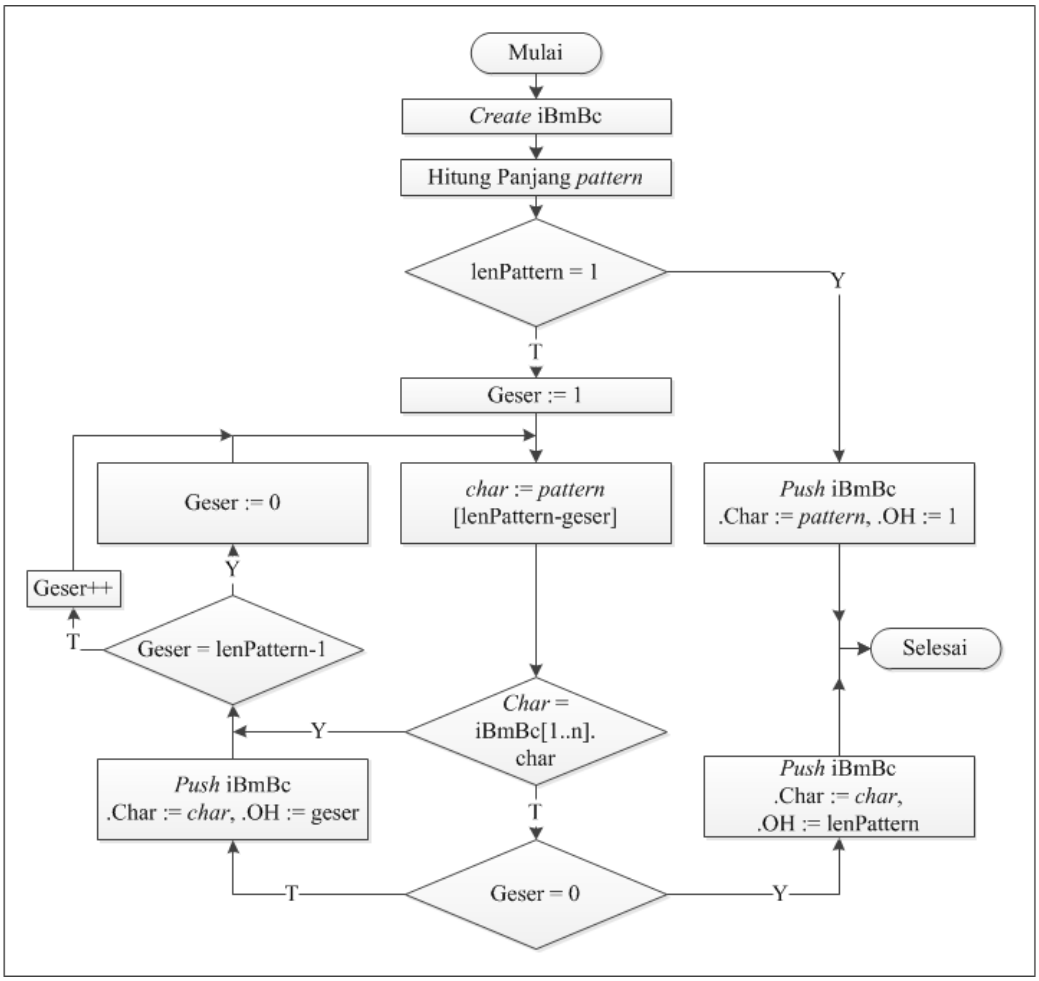
Prosedur preBmBc memiliki tiga nilai penting, diantaranya :

1. *Pattern*, sebagai subjek pencocokan terhadap teks.
2. Karakter, sebagai karakter-karakter yang terdapat pada *pattern*.
3. Occurence Heuristic (OH), sebagai nilai pergeseran yang diperoleh ketika menemukan ketidakcocokan karakter.
4. Pergeseran, sebagai nilai yang dicapai ketika melakukan pergeseran dari kanan ke kiri *pattern*.

Langkah-langkah pelaksanaan prosedur :

1. *Create* atau berikan nilai kosong ke stack iBmBc.
2. Lakukan perhitungan terhadap panjang *pattern*. Jika panjang tidak lebih dari satu, maka hentikan proses dengan menambahkan langsung nilai OH dan karakter ke dalam stack iBmBc. Jika tidak, maka lanjutkan ke proses selanjutnya.
3. Cacahan karakter *pattern* mulai dari karakter ke-2 paling kanan.
4. Bandingkan setiap karakter yang dicacah terhadap stack iBmBc, jika karakter yang dicacah tidak ditemukan di dalam stack, maka tambahkan karakter tersebut ke dalam stack dimana OH sama dengan jumlah pergeseran karakter yang telah dilakukan.
5. Lakukan langkah-4 kembali, dengan melakukan perpindahan 1 karakter ke kiri hingga mencapai karakter paling kiri secara terus menerus.
6. Jika telah mencapai karakter paling kiri, cacah karakter paling kanan lalu kembali ke langkah-4.

Berikut representasi prosedur preBmBc dengan menggunakan flowchart :

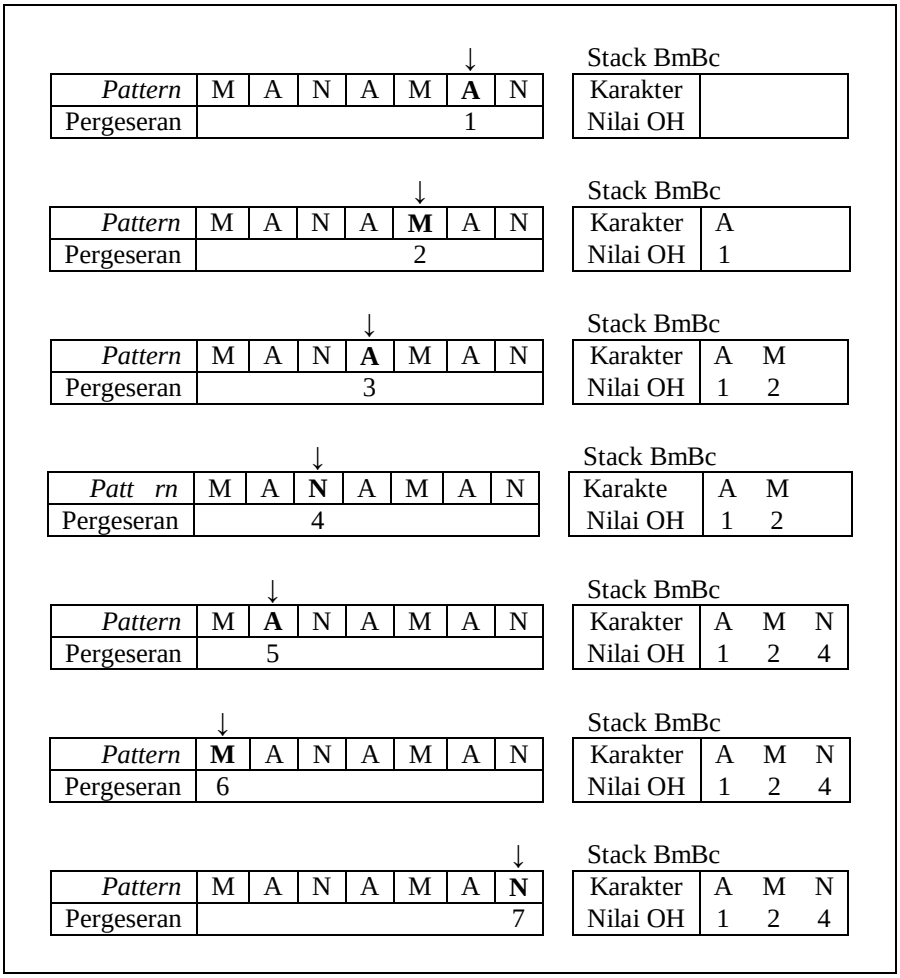


Gambar 4.4 Flowchart pengembangan prosedur preBmBc

Contoh kasus,

Pattern : MANAMAN

Penyelesaian :



Gambar 4.5 Penyelesaian contoh kasus Prosedur preBmBc

Pada gambar di atas, terlihat bahwa karakter yang ditunjuk akan ditambahkan ke dalam stack BmBc jika tidak ditemukan pada stack BmBc dengan nilai OH yang berkesesuaian dengan nilai pergeseran.

4.3.1.2. Prosedur preBmGs

Prosedur preBmGs memiliki enam nilai penting, diantaranya:

1. *Pattern*, sebagai subjek pencocokan terhadap teks.
2. Match Heuristic (MH), sebagai nilai pergeseran yang diperoleh ketika menemukan kecocokan suffix.

3. Compare, sebagai akhiran atau sejumlah karakter sebelah kanan dari sebuah karakter *pattern* yang diperoleh dari pergeseran kanan ke kiri.
4. *Prefix*, sebagai awalan atau karakter *pattern* yang diperoleh dari pergeseran dari kiri ke kanan.
5. *Suffix*, sebagai akhiran sebelah kanan *prefix.*
6. Pergeseran, sebagai nilai yang dicapai ketika melakukan pergeseran dari compare.

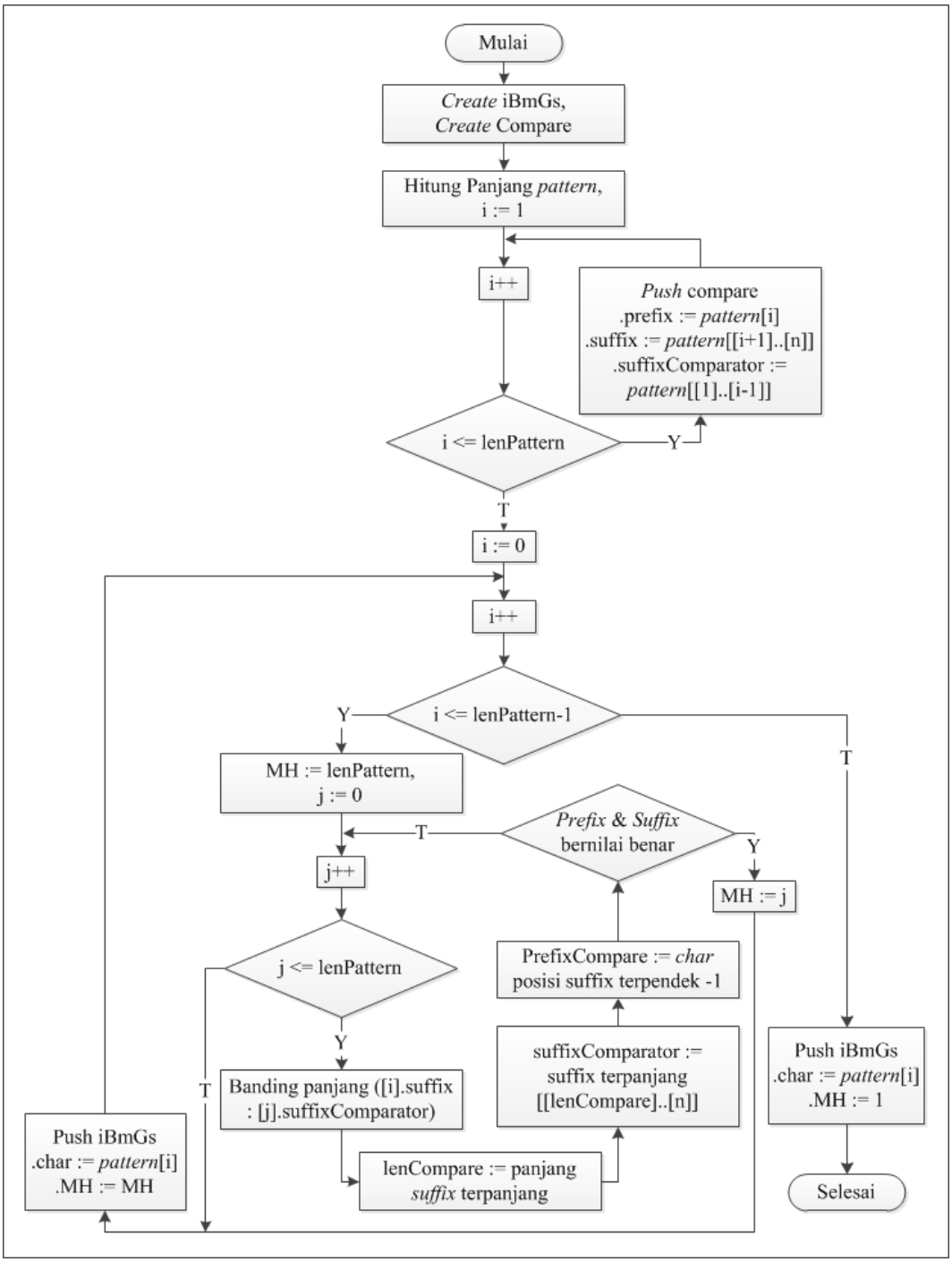
Langkah-langkah pelaksanaan prosedur :

1. Create atau berikan nilai kosong ke stack iBmGs dan Compare.
2. Lakukan aksi yang sama dengan preBmBc jika panjang *pattern* tidak lebih dari satu.
3. Cacah compare terhadap *pattern*. Lalu simpan hasil pencacah dengan menambahkan ke dalam stack compare.
4. Cacah *prefix* dan *suffix* terhadap *pattern.* Lalu simpan hasil pencacahan *prefix* ke dalam stack BmGs, dan bandingkan *suffix* terhadap compare dengan melihat kecocokan antara masing-masing *prefix.*
5. Jika panjang kedua *suffix* tidak sama besar, cacah salah satu *suffix* yang memiliki panjang terbesar kearah kanan sepanjang *suffix* dengan panjang terkecil. Jika ditemukan kecocokan antar *suffix*, bandingkan *prefix* kedua *suffix*. Jika *prefix* yang diperbandingkan adalah sama, maka kecocokan tidak dapat diterima.
6. Untuk compare, *prefix* didapat ketika terjadi pencacahan pada compare, *prefix* akan bernilai sama dengan karakter paling kanan dari hasil pencacahan

terhadap compare sebelum terjadinya pencacahan. Jika tidak terjadi pemotongan, maka *prefix*-compare bernilai null.

7. Untuk indeks *prefix* paling kanan, nilai MH harus selalu bernilai 1. Sedangkan nilai MH pada prefix lainnya, diberikan ketika ditemukan ketidakcocokan antara *prefix-suffix* terhadap *prefix*-compare dan ditemukannya kecocokan di antara kedua *suffix* sebesar nilai pergeseran terbentuknya compare.
8. Ulangi langkah 5 hingga compare yang terakhir.
9. Ulangi langkah 5 hingga *suffix* yang terakhir.

Berikut representasi prosedur preBmGs dengan menggunakan flowchart :



Gambar 4.6 Flowchart pengembangan prosedur preBmGs

Contoh kasus,

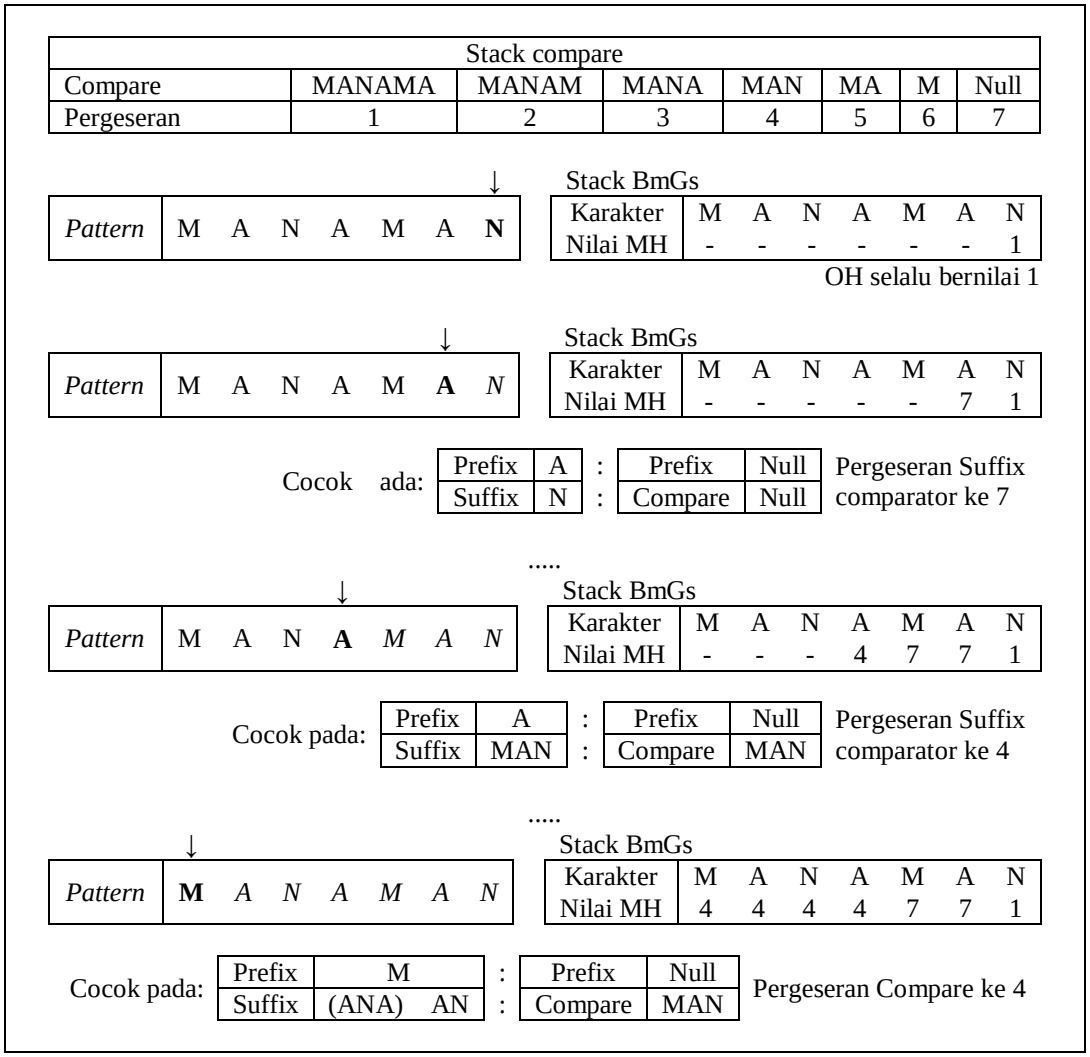
Pattern : MANAMAN

Penyelesaian :

Tabel 4.1 Daftar pencacahan *prefix* dan *suffix* dari *pattern*

Suffix (kanan-kiri)	*Prefix*	*Suffix* (kiri-kanan)
Null	M	ANAMAN
M	A	NAMAN
MA	N	AMAN
MAN	A	MAN
MANA	M	AN
MANAM	A	N
MANAMA	N	Null

Dari tabel di atas, diasumsikan bahwa telah terjadi pemotongan pada kedua sisi *suffix* terhadap *prefix* pada *pattern*. Terlihat bahwa masing-masing *suffix* dipersatukan oleh *prefix* hingga menjadi satu kesatuan *pattern* utuh. Sedangkan null diartikan kosong, yakni semua kondisi pencocokan akan bernilai benar jika dilakukannya perbandingan terhadap karakter tersebut.



Gambar 4.7 Penyelesaian contoh kasus Prosedur preBmGs

Pada gambar di atas terlihat bahwa pergeseran pada compare memiliki peran utama dalam penentuan nilai MH. Berbeda dengan nilai OH, nilai MH selalu diberikan pada karakter-karakter yang ada pada *pattern.* Penyusunan stack BmGs pun berdasarkan urutan indeks posisi karakter pada *pattern*.

4.3.1.3. Prosedur BM

Prosedur BM memiliki empat nilai penting, diantaranya :

1. *Pattern*, merupakan subjek pencocokan terhadap teks.

2. Teks, sebagai objek pencocokan.
3. Stack BmBc, sebagai pembanding ketika ditemukan ketidak cocokan.
4. Stack BmGs, sebagai pembanding ketika ditemukan kecocokan.

Langkah-langkah pelaksanaan prosedur :

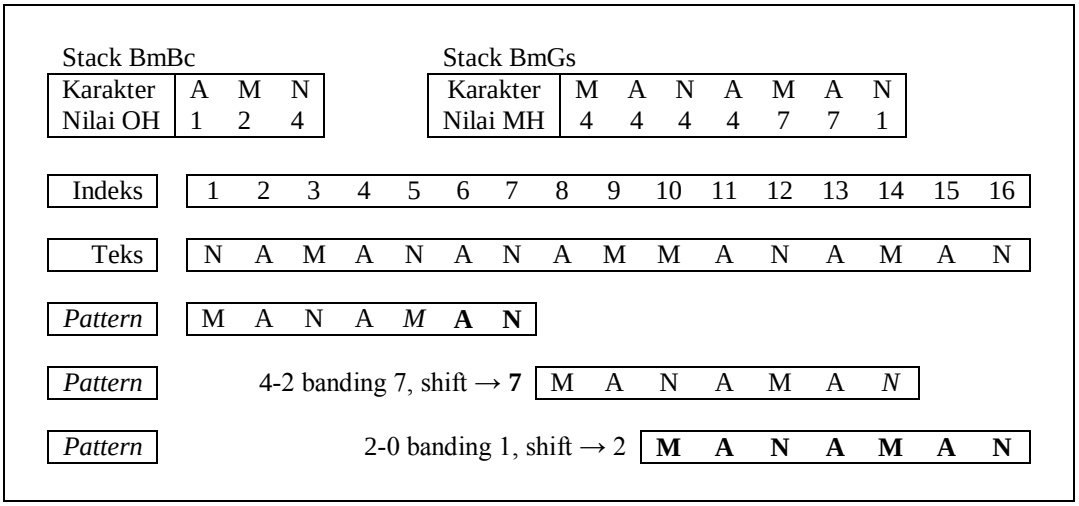
1. Pencocokan dimulai dari indeks teks terkecil atau dari karakter paling kiri.
2. Pencocokan per karakter dimulai dari karakter paling kanan *pattern.*
3. Setiap kali ditemukan ketidakcocokan karakter, maka ambil nilai OH pada stack BmBc dengan karakter yang berkesesuaian terhadap teks, lalu nilai OH dikurangi dengan jumlah kecocokan yang telah terjadi. Ambil nilai MH pada stack BmGs dengan indeks karakter yang ditemukan ketidakcocokan, lalu bandingkan dengan nilai OH yang telah dioperasikan. Dengan nilai terbesar, geser *pattern* kearah kiri teks dan ulangi pencocokan.
4. Jika nilai kecocokan sama dengan panjang *pattern* maka *pattern* telah ditemukan pada teks, geser *pattern* sepanjang jumlah karakter *pattern* ke arah kiri teks untuk melanjutkan pencocokan selanjutnya.

Contoh kasus,

Pattern : MANAMAN

Teks : NAMANANAMMANAMAN

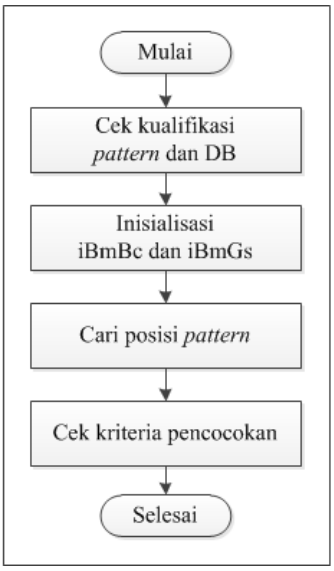
Penyelesaian :



Gambar 4.8 Penyelesaian contoh kasus Prosedur BM

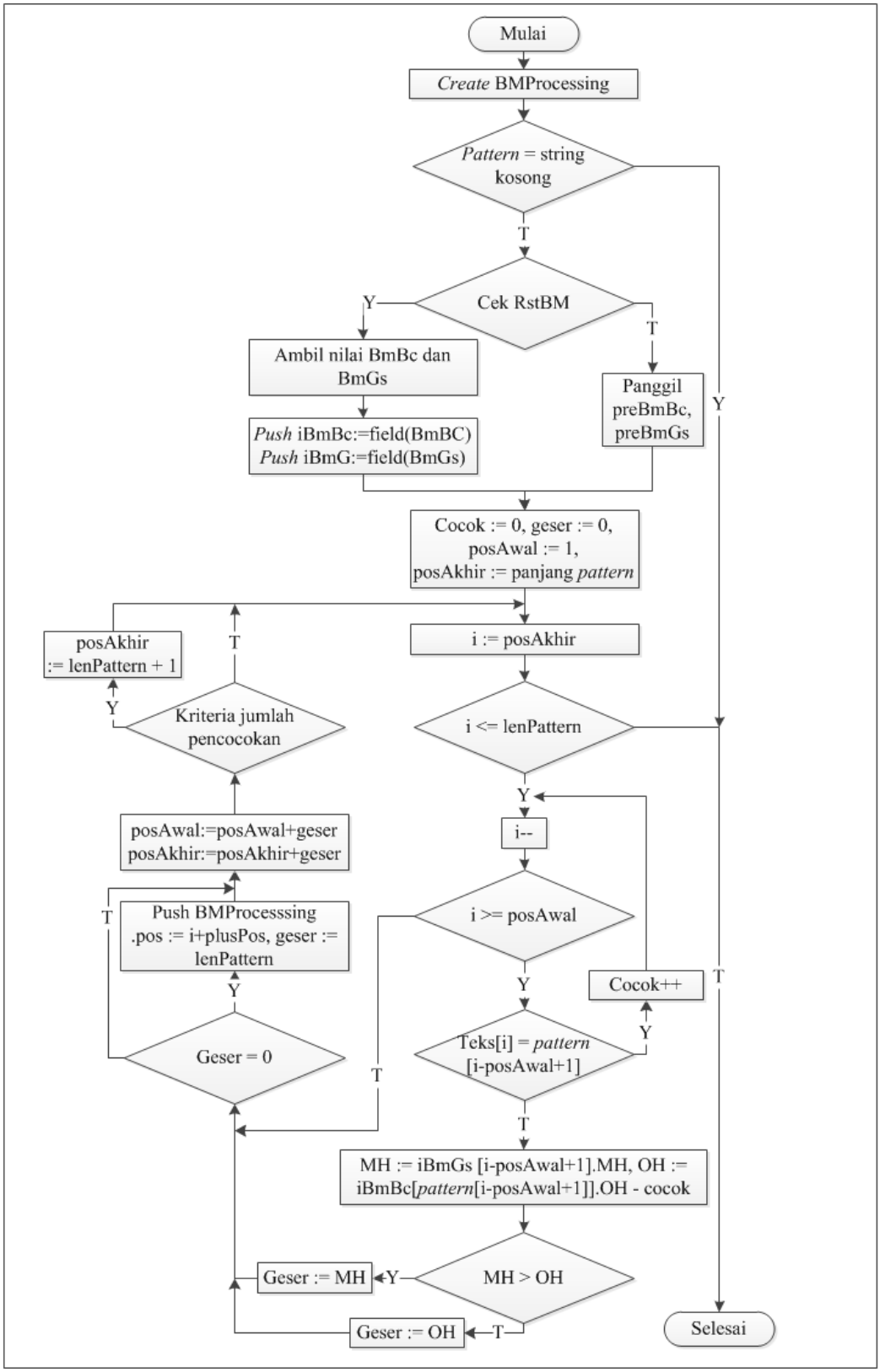
Pada gambar di atas terlihat bahwa pergeseran pada *pattern* selalu mengikuti keputusan dari hasil perbandingan OH dan MH. Pada umumnya, pencocokan standar akan mencocokan karakter per karakter, ketika ditemukan ketidakcocokan maka *pattern* akan digeser satu langkah ke arah selanjutnya. Namun tidak demikian pada algoritma Boyer-Moore. Inilah yang menjadi alasan utama penulis menerapkan algoritma ini sebagai pusat pemecahan pernyataan Pascal sebagai awal proses penerjemahan dan ketika dibutuhkannya sebuah proses pencocokan string.

Berikut detail pengembangan prosedur BM sebagai pencapaian proses pencocokan dan validasi string masukan agar menjadi lebih efisien terhadap waktu proses.



Gambar 4.9 Flowchart pengembangan algoritma Boyer-moore

Terlihat pada gambar bahwa terdapat penambahan proses pada algoritma Boyer-Moore, yaitu cek kualifikasi *pattern* dan database, serta terdapat proses kualifikasi pencocokan. Berikut detail pengembangan proses algoritma Boyer-Moore.



Gambar 4.10 Flowchart detail pengembangan algoritma Boyer-moore

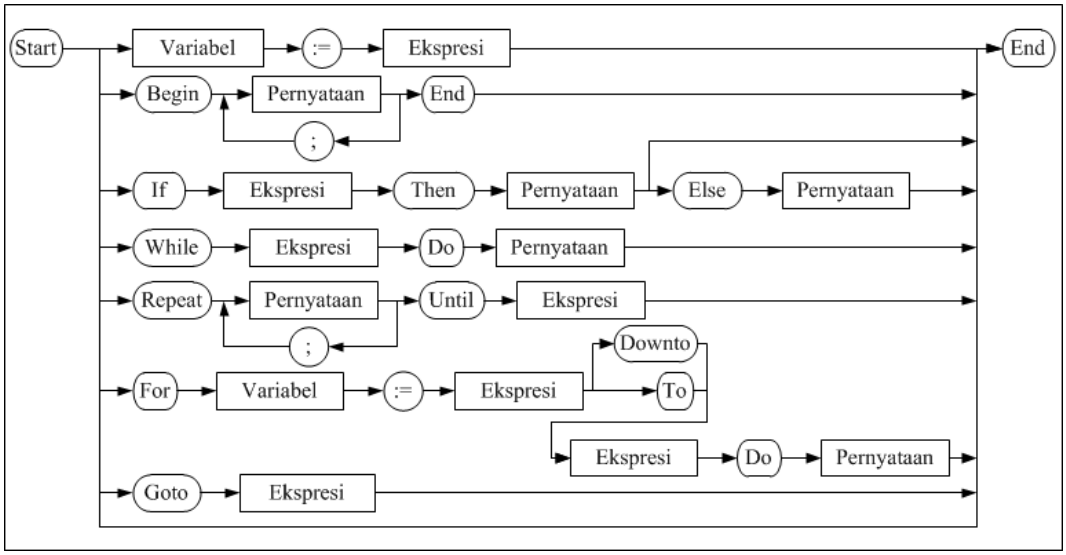
Proses pengambilan nilai BmBc dan BmGs pada database dimaksudkan agar proses perhitungan nilai keputusan pergeseran oleh prosedur BmBc dan BmGs dapat menjadi lebih optimal, karena nilai keputusan telah tersimpan untuk beberapa *pattern* dengan nilai ketetapan. Namun proses filter pada database (recordset) membutuhkan estimasi waktu tertentu. Maka jika dibandingkan proses pencocokan nilai keputusan oleh prosedur preBmBc dan preBmGs untuk *pattern* yang lebih pendek, proses cek database akan menjadi lebih lama. Untuk itu, keputusan pengecekan database ditentukan dari parameter prosedur BM pada saat pemanggilan.

Terdapat pula parameter plusPos yang berfungsi sebagai nilai tambah untuk setiap posisi yang ditemukan. plusPos digunakan pada saat pencocokan string yang membutuhkan posisi akhir, tengah, maupun posisi tertentu dari sebuah karakter yang terdapat pada pattern terhadap teks.

Terdapat pula kriteria jumlah pencocokan, kriteria ini difungsikan sebagai parameter jumlah pencocokan yang dibutuhkan. Sebagai contoh, jika terdapat 10 kecocokan, maka pencocokan akan dihentikan ketika mencapai kecocokan ke-5, dimana 5 adalah kriteria jumlah pencocokan. Sehingga menjadikan algoritma Boyer-Moore berjalan lebih optimal dan sesuai kebutuhan Translator.

4.3.2. Basis Pengetahuan Algoritma Translator

Basis pengetahuan algoritma Translator Pascal ke C dibangun oleh 2 dasar sintaksis seperti yang telah dijelaskan sebelumnya. Berikut representasi sintaksis pemenggalan teks berdasarkan sintaksis pernyataan Pascal.



Gambar 4.11 Flowchart Sintaksis Pernyataan Bahasa Pascal

Pada gambar di atas, terlihat bahwa masing-masing pernyataan memiliki alur yang bernilai tetap. Alur-alur tersebut membentuk sebuah pola pembentukan pernyataan yang dapat dimanfaatkan oleh algoritma Translator untuk mengkondisikan kriteria pencocokan *pattern* algoritma Boyer-Moore terhadap teks. Terlihat pula bahwa suatu pernyataan dapat memiliki badan berupa pernyataan lainnya, dan tentunya hal ini berdasarkan pengkondisian pernyataan induk. Maka dengan mengaitkan per pola variasi pernyataan Pascal terhadap C, didapatlah suatu aturan baru mengenai pemenggalan dan penggabungan hasil dari pencocokan algoritma Boyer-Moore terhadap teks.

Dengan pemanfaatan Database Microsoft Accsess 2003 sebagai media penyimpanan data, pengkondisian pola sintaksis dirancang dan disimpan berdasarkan keterkaitan yang telah dijelaskan pada pembahasan sebelumnya. Sehingga dapat digunakan oleh Translator sebagai basis pengetahuan pemberian *pattern* pencocokan algoritma Boyer-Moore dan menvalidasi nilai posisi yang

dihasilkan. Sehingga didapatkan nilai posisi pemenggalan suatu teks berdasarkan sintaksis Pascal, dan penggabungan teks berdasarkan sintaksis C.

Dalam penggunaannya, Translator menggunakan enam query sebagai Control Recordset Visual Basic 6.0 untuk mendukung dan meningkatkan proses pencocokan dan penerjemahan.

Berikut daftar query algoritma Translator:

1. Query rstLibrary, terdiri dari tabel-tabel yang mengkondisikan hubungan antara pernyataan Pascal dan C.

Tabel 4.2 Fungsionalitas Query rstLibrary

Nama Field	Fungsi
Pascal	Bentuk awal pengenal
C	Bentuk akhir / terjemahan pengenal
AnyExpression	Nilai keharusan kepunyaan ekspresi
NonExpression	Nilai keharusan ketidakpunyaan ekspresi
ExpContent	Kombinasi keharusan penggunaan variabel, teks/string, dan string kontrol
VarList	Nilai keharusan penggunaan variabel
SymbolList	Nilai keharusan penggunaan string kontrol
TextList	Nilai keharusan penggunaan string/teks
SpecialCondition	Nilai kondisi khusus pengenal
SuffixFormat	Akhiran/format tutupan pengenal

2. Query rstDataType, terdiri dari tabel-tabel yang menginformasikan keterkaitan tipe data antara Pascal dan C.

Tabel 4.3 Fungsionalitas Query rstDataType

Nama Field	Fungsi
BasicType	Bentuk dasar awal tipe data
PascalType	Bentuk awal tipe data variasi
Range	Informasi jangkuan tipe data variasi
Size	Informasi ukuran tipe data variasi
Level	Prioritas tipe data variasi
Ctype	Bentuk akhir tipe data variasi

Tabel 4.3 Fungsionalitas Query rstDataType (lanjut)

Nama Field	Fungsi
CSymbol	String kontrol tipe data variasi

3. Query rstOperationResult, terdiri dari tabel-tabel yang mengiformasikan nilai keharusan tipe data hasil dari suatu operasi ekspresi.

Tabel 4.4 Fungsionalitas Query rstOperationResult

Nama Field	Fungsi
POperator	Bentuk awal operator
COperator	Bentuk akhir operator
BasicType1	Tipe data dasar operand 1
BasicType2	Tipe data dasar operand 2
BasicType3	Tipe data dasar hasil operasi

4. Query rstBM, terdiri atas tabel BM yang berisi komponen utama algoritma Boyer-Moore, sehingga untuk penggunaan beberapa *pattern*, algoritma Boyer-Moore tidak perlu melakukan pendefinisian OH dan MH. Dan tentunya, basis pengetahuan ini akan mempercepat proses pencocokan algoritma Boyer-Moore.

Tabel 4.5 Fungsionalitas Query rstBM

Nama Field	Fungsi
Pascal	*Pattern*
BmBc	Nilai OH
BmGs	Nilai MH

5. Query rstOperators, terdiri atas tabel operator yang difungsikan sebagai *pattern* disaat menjalankan proses penerjemahan operasi ekspresi.

Tabel 4.6 Fungsionalitas Query rstOperators

Nama Field	Fungsi
POperator	Bentuk awal operator
OperatorDes	Informasi Operator
COperator	Bentuk akhir operator
Space	Nilai keharusan karakter spasi
Hierarchy	Hirarki operator

6. Query rstEKeyword, terdiri atas tabel ExceptionKeyword yang difungsikan sebagai aturan khusus dalam proses pengenalan teks dan pengeneralisasian sintaksis Pascal untuk beberapa pengenal dengan aturan penulisan yang cenderung memiliki kekhususan tersendiri.

Tabel 4.7 Fungsionalitas Query rstEKeyword

Nama Field	Fungsi
PEKeyword	*Pattern* target
EConvert	*Pattern* subsitusi/pengganti

4.3.3. Teks Sebagai Pernyataan Pascal

Suatu teks akan dikenali sebagai pernyataan Pascal oleh Translator jika memiliki tiga bagian utama dari pembentukan sebuah pernyataan Pascal. Translator mendefinisikannya sebagai *Left-string*, *Middle-string*, dan *Right-string*. Ketiga bagian inilah yang menentukan kebenaran pernyataan secara struktur pembentukan pernyataan pascal terhadap hasil penerjemahan Pascal ke C. Apabila terjadi keganjilan dalam proses penerjemahan, maka secara otomatis Translator akan menyisipkan sebuah kode diposisi teks yang bermasalah. Kode-kode penyisipan ini akan menjadi *final step* terhadap pemberian pesan kesalahan.

4.3.3.1. Pemenggalan Teks

Berikut representasi proses pemenggalan teks yang mempunyai 100% nilai kebenaran teks sebagai pernyataan pascal, dimana diasumsikan varA bertipe real, varB bertipe string, dan i, j, k bertipe integer.

```
RePeat
        Write (varA, varB, 'pAsCal');
UntIl (i * (j + k)) > 5;
Read (varA);
varB:=varA+7;
```

Gambar 4.12 Format teks asli

4.3.3.1.1. Atur Ulang Teks

Fungsi ini ditujukan untuk memanipulasi beberapa kata yang memiliki aturan khusus penulisan agar dapat dikenali saat proses pemenggalan teks menjadi pernyataan tunggal.

1. Ubah input menjadi format *default* yang dapat dikenali oleh algoritma Translator dan Boyer-Moore, yaitu merubah setiap karakter huruf menjadi huruf kecil pada teks dan tambahkan satu karakter titik koma di awal dan akhir teks.
2. Lakukan pencocokan karakter kutip pada teks dengan menambahkan satu karakter kutip di awal dan akhir teks sebagai subjek pencocokan algoritma Boyer-Moore.
3. Jika jumlah karakter kutip sama dengan genap, maka proses dilanjutkan dengan menyimpan seluruh karakter pada area antara kutip buka (indeks posisi kutip genap) dan kutip tutup (indeks posisi kutip ganjil) ke dalam sebuah *stack* dari teks asli. Lalu ganti seluruh kombinasi karakter yang berada

di posisi yang sama dengan nilai separator (rangkaian kombinasi karakter acak sepanjang 10 karakter yang telah dikodekan sebelumnya) pada teks. Jika jumlah karakter kutip sama dengan ganjil maka proses dihentikan.

4. Ambil nilai setiap kolom di query rstEKeyword pada database, lalu lakukan pencocokan dengan nilai kolom PEKeyword sebagai *pattern* terhadap teks.
5. Jika PEKeyword ditemukan, maka cek karakter paling pertama selain karakter *space*, *tab*, dan *newline* yang berada sebelum posisi PEKeyword, jika karakter yang ditemukan sesuai dengan nilai LeftChar dan RightChar pada record yang sama, maka lanjutkan pengecekan terhadap karakter setelah posisi ujung PEKeyword. Jika karakter sesuai, maka ganti setiap karakter yang berada di sepanjang posisi PEKeyword dengan nilai Econvert yang sebelumnya telah dilakukan proses penempelan Separator pada karakter "#" yang terdapat pada Econvert.
6. Lanjutkan pencocokan dengan nilai PEKeywod yang lain.

```
;r[separator]epeat;
        write (vara, varb, [separator]);
;u[separator]ntil (i * (j + k)) > 5;
read (vara);
varb:=vara+7;;
```

Gambar 4.13 Hasil Proses Atur Ulang Teks

4.3.3.1.2. Pemenggalan Teks Menjadi Pernyataan Tunggal

Pada proses ini, teks akan dipecah menjadi pernyataan tunggal sehingga memungkinkan untuk terdefinisikannya bagian pengenal dan ekspresi.

1. Lakukan pencocokan pada teks dengan karakter titik koma sebagai *pattern* pencocokan Boyer-Moore.
2. Potong seluruh teks yang berada di sebelum posisi titik koma dan lanjutkan ke fungsi pemotongan pernyataan tunggal. Lakukan hingga mencapai posisi terakhir titik koma.

```
; ..pernyataan tunggal pertama
r[separator]epeat; ..pernyataan tunggal kedua
write (vara, varb, [separator]); ..pernyataan tunggal ketiga
; ..pernyataan tunggal keempat
u[separator]ntil (i * (j + k)) > 5; ..pernyataan tunggal kelima
read (vara); ..pernyataan tunggal keenam
varb:=vara+7; ..pernyataan tunggal ketujuh
; ..pernyataan tunggal kedelapan
```

Gambar 4.14 Format *single* statatement

4.3.3.1.3. Pemenggalan Pernyataan Tunggal Menjadi *Left-Middle-Right* String

Pada proses ini, pembagian pernyataan tunggal akan dilaksanakan berdasarkan asas penggunaan sebuah ekspresi pada pengenal. Hasil dari pemenggalan ini merupakan kunci utama keberhasilan proses penerjemahan.

1. Berikan nilai *closed-bracket* sebagai panjang pernyataan tanpa karakter titik koma di akhir.

2. Cari posisi karakter buka-kurung sebagai *pattern* terhadap *single* pernyataan, lalu isikan nilai *open-bracket* dari posisi hasil pencocokan. Jika tidak ditemukan, isikan nilai *open-bracket* sebagai posisi setelah *closed-braket*.
3. Cari posisi karakter *Newline*, *Space*, dan := sebagai *pattern* dan single pernyataan sebagai subjek pencocokan. Lalu tentukan kemunculan pertama dari tiga karakter diatas dan bandingkan dengan posisi buka-kurung. Karakter dengan kemunculan pertama akan menjadi posisi *open-bracket*.
4. Potong text yang berada pada area sebelum posisi *open-bracket* sebagai *left-string*, posisi setelah *closed-bracket* sebagai *right-string*, dan sisanya menjadi *middle-string*.

Tabel 4.8 Format *Left-Middle-Right String*

Pernyataan	***Left-string***	***Middle-string***	***Right-string***
1	*[karakter kosong]*	*[karakter kosong]*	;
2	r[separator]epeat	*[karakter kosong]*	;
3	write	(vara, varb, [separator])	;
4	*[karakter kosong]*	*[karakter kosong]*	;
5	u[separator]ntil	(i * (j + k)) > 5	;
6	read	(vara)	;
7	varb	:=vara+7	;
8	*[karakter kosong]*	*[karakter kosong]*	;

4.3.3.2. Penerjemahan *Left-String*

Left-string dimaksudkan sebagai pengenal dari sebuah pernyataan. Setiap jenis pengenal mempunyai hasil terjemahan yang berbeda. Perbedaan inilah yang akan menvalidasi nilai kebenaran dari *middle-string.*

1. Jika *left-string* bernilai kosong – selain karakter *space*, *tab*, *newline* dan *Null*, string dianggap tidak kosong – maka proses penerjemahan dihentikan. Jika tidak lanjutkan ke point berikutnya.

2. Lakukan pencarian posisi separator pada *left-string*, jika ditemukan maka hapus seluruh karakter yang berada di jangkuan area posisi separator,
3. Lakukan proses filterisasi *left-string* terhadap rstLibrary pada kolom pIdentifier. Jika *left-string* tidak ditemukan, maka cek apakah *left-string* berupa variabel yang terdaftar atau tidak dengan bantuan fungsi isVar (prosedur penunjang). Jika berupa variabel, maka ubah format *left-string*, *middle-string*, dan cIdentifier agar dapat dikenali oleh algoritma Translator sebagai sebuah variable dan ekpresi. Lalu proses ulang *left-string*.
4. Jika pencocokan *Left-string* telah selesai, berikan *feedback* sebagai hasil penerjemahan *left-string*, dan lanjutkan penerjemahan *middle-string* dan -*right-string*.

Tabel 4.9 Hasil Proses *Left-string*

Pernyataan	***Left-string***	***Feedback***
1	*[karakter kosong]*	*[karakter kosong]*
2	repeat	Do
3	write	Printf
4	*[karakter kosong]*	*[karakter kosong]*
5	until	While
6	read	Scanf
7	Varb	Proses ulang dengan, *Left-string* ‘varb=’ cIdentifier ‘s#pecialvar’ *middle-string* ‘vara+7’
8	*[karakter kosong]*	*[karakter kosong]*
7	varb=	varb=

4.3.3.3. Penerjemahan *Middle-String*

Middle-string difungsikan sebagai bagian ekspresi dari suatu pernyataan. Dibeberapa pengenal, terdapat perbedaan-perbedaan dalam penggunaan ekspresi.

Untuk itu dibutuhkan beberapa fungsi yang dapat menvalidasi nilai kebenaran sebuah ekspresi terhadap nilai ketergantungan.

1. Proses dimulai dengan membaca aturan yang terdapat pada rstLibrary yaitu pada kolom AnyExpression. Nilai pada AnyExpression akan menentukan apakah *middle-string* telah sesuai format secara utuh / benar, sebagian, atau salah keseluruhan dari sudut pandang penggunaan karakter buka-tutup kurung.
2. Jika hasil pembacaan AnyExpression menyatakan salah keseluruhan, maka proses penerjemahan dihentikan. Jika terdapat posisi buka-tutup kurung yang tidak normal, maka lakukan penyisipan kode kesalahan pada *middle-string*. Jika normal, maka dilanjutkan ke tahap selanjutnya.
3. Selanjutnya melakukan cek terhadap penggunaan karakter koma atau struktur global dari sebuah ekspresi Pascal, yaitu penggunaan operator “,” pada runtunan operasi pada ekspresi. Jika terjadi kesalahan, maka proses dihentikan. Jika benar, maka dilanjutkan ke proses pengecekan struktur lokal dari sebuah ekspresi Pascal.
4. Struktur lokal mempunyai dependensi terhadap *left-string*. Dengan adanya ketergantungan ini, maka aturan pada struktur lokal dapat diambil di rstLibrary yang sudah melalui proses filter sebelumnya. Struktur lokal menyatakan bahwa suatu sintaks harus mempunyai jenis ekspresi dengan kategori variabel, teks/string, angka ataupun kombinasinya. Dengan penambahan sintaksis C di aturan ini, maka didapat aturan lokal mengenai simbol tipe data hasil ekspresi. Adanya aturan sintaksis ini, maka pengecekan

middle-string akan diteruskan sesuai jenis kategori yang dinyatakan. Jika benar maka lanjut ke proses selanjutnya, jika salah maka proses dihentikan.

5. Proses selanjutnya dengan melakukan operasi terhadap teks *middle-string* untuk mendapatkan simbol tipe data yang dihasilkan dengan merubah ekspresi menjadi *single*-ekspresi. *Single*-ekspresi didefinisikan sebagai ekspresi yang terpisah oleh karakter koma. Dari hasil operasi *single*-ekspresi inilah simbol tipe data ekspresi dapat diketahui dan dihasilkan pula sebuah ekspresi baru dengan format bahasa C. Pengoperasian ekpresi didukung oleh beberapa fungsi yang melakukan proses penyamaan sintaksis antara Pascal dan C yang dihubungkan melalu control rstOperators dan rstOperationResult.
6. Setelah operasi ekspesi selesai, maka dilakukan pembacaan aturan penyususnan antara *single*-ekspresi Pascal dan C. Untuk beberapa pengenal, format penyusunan memiliki sebuah struktur spesial. Struktur spesial inilah yang menjadi validasi akhir proses penerjemahan *middle-string.*

4.3.3.4. Penerjemahan *Right-String*

Karakter yang masuk ke proses ini selalu berupa karakter titik-koma. Hal tersebut disebabkan oleh pengaturan ulang teks oleh algoritma Translator dengan bantuan algoritma Boyer-Moore. Pengaturan ulang dilakukan karena dalam hubungan sintaksis antara Pascal dan C, akhiran dari suatu sintaks C tidak selalu berkodekan titik koma. Untuk itulah proses ini berperan dalam menentukan karakter yang akan diberikan pada akhir pernyataan.

Langkah penerjemahan pada *right-string* adalah dengan mengambil nilai kolom SuffixFormat rstLibrary yang sudah difilter sebelumnya terhadap *left-string* maka proses penerjemahan hanya menggantikan *right-string* dengan nilai yang terkait sebagai *feedback* proses.

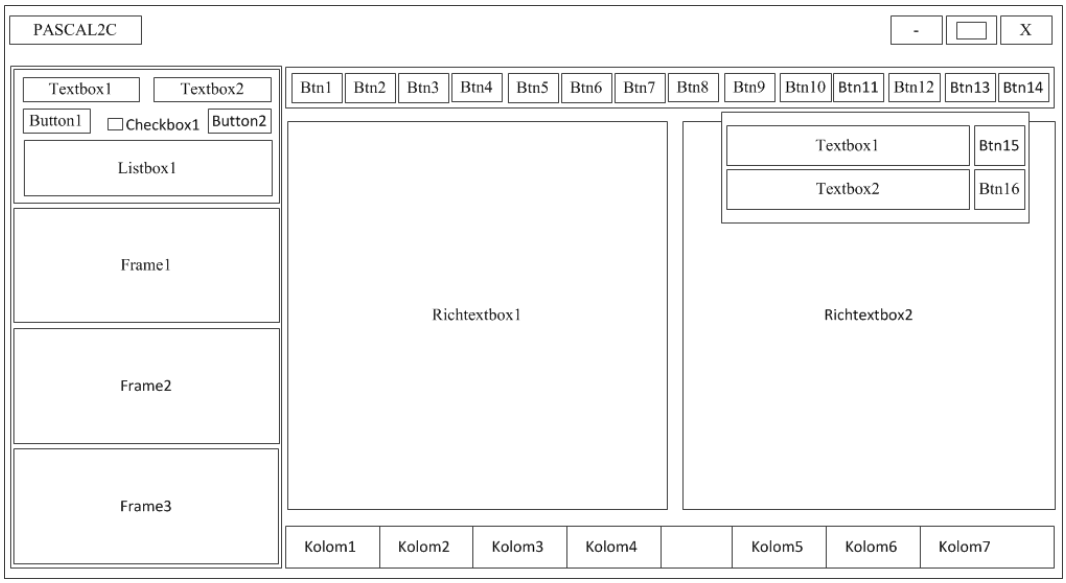
Tabel 4.10 Feedback Proses *Left-string*

Pernyataan	***Feedback***
1	none
2	none
3	;
4	None
5	;
6	; fflush(stdin);
7	;
8	none

4.4. Rancangan Antarmuka Pengguna

Hasil akhir dari Translator yaitu berupa editor teks dengan fungsi-fungsi standar. Perbedaanya, pada editor Translator terdapat fungsi penerjemahan masukan terhadap masukan pengguna dan terdapat *slidebar* yang menginformasikan beberapa kebutuhan dari Translator. Berikut perancangannya :

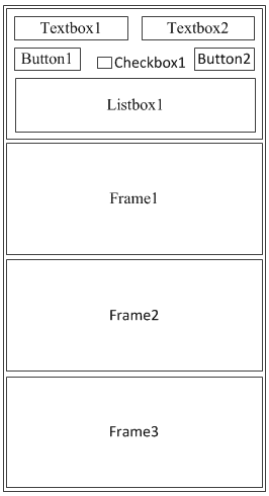
4.4.1. Translator Mode Ruang Edit Standar



Gambar 4.15 Rancangan antarmuka Translator mode ruang edit standar

Gambar di atas merupakan tampilan antarmuka yang dimunculkannya semua fitur Translator pada layar. Richtexbox1 merupakan tempat masukan pengguna, sedangkan richtextbox2 sebagai tempat keluaran hasil terjemahan.

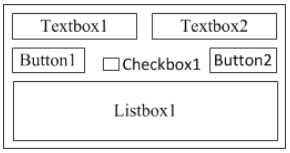
4.4.1.1. Sliderbar



Gambar 4.16 Rancangan antarmuka Slidebar

Pada rancangan, akan ditampilkannya beberapa kebutuhan proses terjemahan yang harus diketahui oleh pengguna dan kebutuhan lainnya yang dapat membantu kelancaran dalam penggunaan aplikasi.

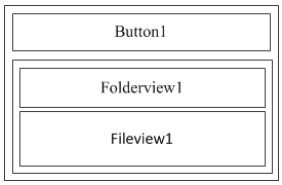
4.4.1.1.1. Ruang Masukan Deklarasi Variabel



Gambar 4.10 Rancangan antarmuka ruang masukan deklarasi variabel

Ruang di atas difungsikan sebagai media pengguna untuk mendeklarasikan variabel-variabel yang akan digunakan pada kode sumber. Ruang input menggunakan textbox1 dan 2 sebagai ruang variabel dan tipe data, sedangkan button1, checkbox1, dan button2 sebagai fungsi tambah dan hapus untuk setiap variabel yang harus terdaftar. Listbox1 merupakan himpunan variabel-variabel yang telah dinyatakan terdeklarasi dengan benar.

4.4.1.1.2. Ruang Kerja / *Workspace*

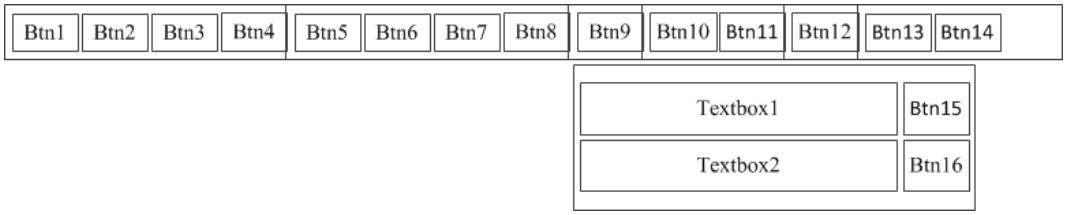


Gambar 4.17 Rancangan ruang kerja

Ruang kerja difungsikan sebagai tempat pengguna untuk melakukan *tracking* terhadap dokumen-dokumen teks yang akan digunakan sebagai masukan. Ruang kerja akan menunjukkan direktori folder yang aktif sebagai sumber

masukan file teks. Tracking ruang kerja akan tampak pada control folderview1, sedangkan fileview1 akan menampilkan isi folder yang mendukung sebagai dokumen masukan terhadap editor.

4.4.1.2. Rancangan Toolbar Translator



Gambar 4.18 Rancangan toolbar Translator

pada gambar di atas, terdapat 16 tombol dan 2 textbox yang masing-masingnya difungsikan sebagai fungsi umum yang terdapat pada sebuah editor teks. Salah satu tombol-tombol tersebutlah yang akan digunakan sebagai pemanggilan fungsi penerjemahan.

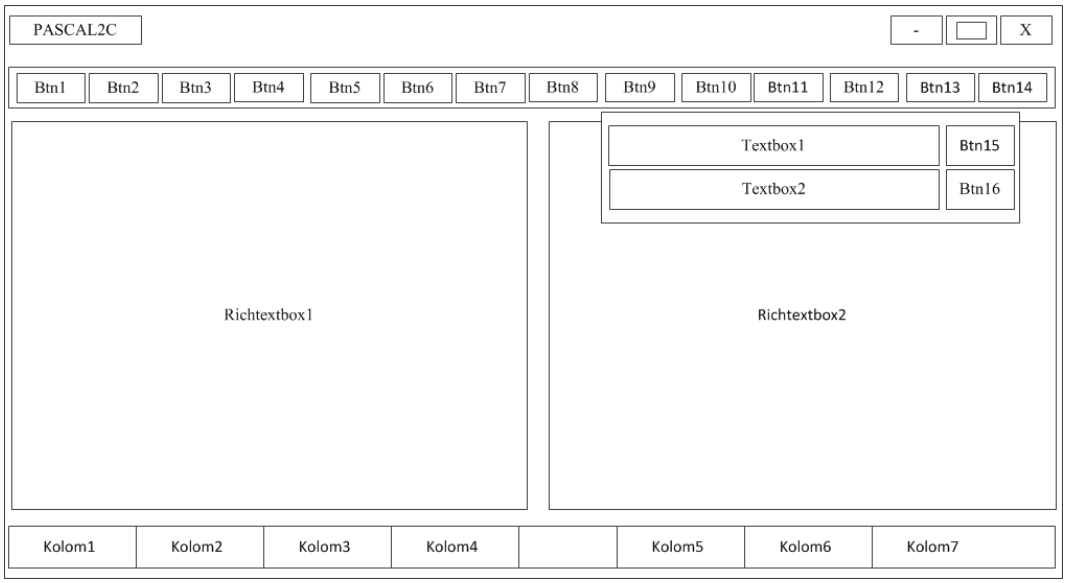
4.4.1.3. Rancangan Messagebar Translator



Gambar 4.19 Rancangan messagebar Translator

Rancangan antar muka ini difungsikan untuk menampilkan pesar-pesan mengenai proses terjemahan. Sebagai contoh, lama waktu yang butuhkan untuk menterjemahkan suatu masukan. Pesan tersebut akan tertampil di messagebar sebagai informasi untuk pengguna.

4.4.2. Translator Mode Ruang Edit Penuh



Gambar 4.20 Rancangan antarmuka Translator mode ruang edit penuh

Rancangan ini dimaksudkan untuk menyediakan ruang yang lebih besar dalam pemasukan teks. Mode ini menghilangkan sliderbar sehingga pengguna dapat lebih leluasa menggunakan ruang editor translator.