PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk *Multi Language Programming*)

PUNIEDIT SOFTWARE DEVELOPMENT
(Software Editor for Multi Language Programming)

Budiman
(budiman1982@gmail.com)

ABSTRACT

During this period continued to develop computer software, programming language was no exception. At the start of the era of low level programming languages, then developed a high level programming language. It is characterized by the appearance of a method of programming offered by a programming language, that is, object-oriented programming (OOP). IDE (Integrated Development Environment) is a computer program that has some facilities that are required in the development of the software. The purpose of the IDEA is to provide all the necessary utilities in building software. As for the type of software text editor that can be used to manipulate the source code hereinafter referred to as the source code of programming languages such as Ultraedit, JediEdit, ClearEdit, cEdit, the Golden Pen, and so on. PuniEdit software is a text-based editor software that can simplify the user through correction, insertion, and modification of the source code.

PuniEdit software is built using Borland Delphi 7.0 and SynEdit component. This software can be used for the Pascal programming language, C++ and HTML. In addition, the software PuniEdit can perform management of the token.

This PuniEdit software, the user can clearly see every occurrence of the type of token as keywords (reserved word), identifier, operator, and so on.

Keywords:

Source code, programming language, source code is scanned.

1. Pendahuluan

Pada masa ini *software* komputer terus berkembang, tak terkecuali bahasa pemrograman. Di mulai era bahasa pemrograman tingkat rendah, kemudian berkembang bahasa pemrograman tingkat tinggi. Hal ini ditandai dengan kemunculan metode pemrograman yang ditawarkan oleh bahasa pemrograman, yaitu pemrograman berorientasi objek (OOP). Karena begitu banyaknya jenis-jenis bahasa pemrograman, bahasa-bahasa tersebut juga dikelompokkan berdasarkan kriteria tertentu. Ada yang mengelompokkannya menjadi 3 level bahasa yaitu: *high level* (Seperti Pascal dan Basic), *middle level* (Seperti Bahasa C), dan *low level* (Seperti Bahasa Assembly). Ada juga yang mengelompokkan berdasarkan paradigma, yaitu *procedural/functional programming*, *object oriented programming*, dan sebagainya.

IDE (*Integrated Development Environment*) merupakan program komputer yang memiliki beberapa fasilitas yang diperlukan dalam pembangunan perangkat lunak. Tujuan dari IDE adalah untuk menyediakan semua utilitas yang diperlukan dalam membangun perangkat lunak. Namun dibalik semua itu kemampuan IDE sendiri membutuhkan memori yang cukup besar, sehingga waktu yang diperlukan untuk menunggu proses loading IDE pun cukup lama. Semua ini tergantung dari komputer yang digunakan.

Text Editor merupakan perangkat lunak editor berbasis teks, dengan *text editor* pemakai dapat melakukan penyuntingan ataupun pengolahan teks. Banyak sekali jenis perangkat lunak *text editor* yang sering digunakan untuk mengolah teks diantaranya adalah Notepad, WordPad, Pico, dan sebagainya. Adapun jenis perangkat lunak *text editor* yang dapat digunakan untuk mengolah *source code* yang selanjutnya disebut kode sumber dari bahasa pemrograman diantaranya adalah Ultraedit, JediEdit, ClearEdit, cEdit, Golden Pen, dan sebagainya.

Perangkat lunak di atas merupakan perangkat lunak *text editor* yang mampu melakukan penyuntingan atau pengolahan kode sumber, dengan menggunakan perangkat lunak *text editor* ini pemakai dapat melihat dengan jelas setiap kemunculan token seperti *keywords* (*reserved word*), *identifier*, *operator*, dan sebagainya.

Berawal dari perangkat lunak *text editor* di atas, penulis ingin membangun suatu perangkat lunak *text editor* yang lebih sederhana. Perangkat lunak yang akan dibangun adalah perangkat lunak PuniEdit.

1.1. Identifikasi Masalah

Bagaimana perangkat lunak PuniEdit yang dibangun dapat mengenali bahasa pemrograman? Perangkat lunak PuniEdit dapat melakukan pendeteksian terhadap file yang berisi kode sumber, yaitu dengan mengidentifikasi ekstension atau tipe dari file tersebut. Dengan teridentifikasinya tipe file maka perangkat lunak PuniEdit dapat mengenali bahasa yang digunakan pada kode sumber tersebut.

1.2. Tujuan

Membangun perangkat lunak PuniEdit yang dapat mengenali bahasa pemrograman dengan memberikan *highlight* atau efek warna pada setiap token yang ditemukan.

1.3. Batasan Masalah

Pada penelitian ini banyak sekali bahasa pemrograman. Sehingga untuk lebih terarah dalam penyusunan penelitian ini, maka dibatasi sebagai berikut:

- a. Bahasa pemrograman yang dapat dikenali oleh perangkat lunak *text editor* PuniEdit, contohnya: Object Pascal, C/C++, HTML.
- b. Hanya mengelola token dari bahasa pemrograman tersebut.
- c. Tidak melakukan analisis sintak dan semantik.

1.4. Metode Pemecahan Masalah

Seiring dengan berkembangnya ilmu dan teknologi informasi yang semakin pesat, pada saat ini banyak metode baru yang dapat digunakan untuk memecahkan persoalan yang berhubungan dengan perangkat lunak. Oleh karena itu pengembangan sistem serta pembangunan perangkat lunak juga tidak dapat lepas dari berbagai metodologi.

Beberapa metodologi pembangunan perangkat lunak dikembangkan orang. Masing-masing metodologi memiliki kelebihan dan kekurangan. Walaupun demikian setiap pendekatan yang ada berusaha agar dapat menghasilkan suatu perangkat lunak yang baik dan sesuai dengan kebutuhan pengguna.

Metode pemecahan yang akan digunakan untuk pembangunan perangkat lunak ini adalah metode *classic life cycle* atau model sekuensial linear atau bisa juga disebut

model waterfall. Siklus ini bersifat sistematis, dengan pendekatan sekuensial untuk membangun perangkat lunak, dimulai pada tingkat sistem berkembang sampai dengan pengujian, dan perbaikan kesalahan.

Metode *classic life cycle* adalah paradigma rekayasa perangkat lunak yang paling luas dipakai dan paling tua. Paradigma model ini memiliki tempat yang terbatas namun penting di dalam kerja rekayasa perangkat lunak. Paradigma itu memberikan *template* dimana metode analisis, desain, pengkodean, pengujian dan pemeliharaan bisa dilakukan. Siklus metode *classic life cycle* tetap menjadi model bagi rekayasa perangkat yang paling luas dipakai. Sekalipun memiliki kelemahan, secara signifikan lebih baik daripada pendekatan yang sifatnya sembarang kepada pengembangan perangkat lunak.

2. Konsep Dasar *Text Editor*

Text editor merupakan perangkat lunak untuk melakukan penyuntingan teks. Pertama kali *text editor* tidak menyediakan tampilan dalam bentuk windows. *Text editor* berbeda dengan word processor, *text editor* ini tidak melakukan pengaturan format dokumen atau fitur-fitur lainnya seperti yang digunakan pada *desktop publishing*.

Text editor sering disediakan oleh sistem operasi atau paket dari *software development*, dan dapat digunakan untuk merubah konfigurasi file serta kode sumber dari bahasa pemrograman.

Text editor ini mempunyai ukuran yang kecil dan mudah digunakan, sementara yang lain menawarkan fungsi yang lebih luas dan rumit. Sebagai contohnya, Unix yang mempunyai editor vi, tapi banyak juga yang menggunakan editor Emacs untuk melakukan penyuntingan teks. Microsoft Windows juga mempunyai *text editor* yang lebih sederhana yaitu Notepad.

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

Di bawah ini merupakan contoh perangkat lunak *text editor*, yaitu
[http://en.wikipedia.org/wiki/Comparison_of_text_editors]:

Tabel 2.1 Text Editor

No	Nama Text Editor	Produsen	Lingkungan Sistem Operasi			
			Windows	Linux	Unix	Mac OS
1.	Acme	Rob Pike	Ya	Ya	Ya	Ya
2.	Alpha	Pete Keleher, now the Alpha-development cabal	Tidak	Tidak	Tidak	Ya
3.	Alphatk	Vince Darley	Ya	Ya	Ya	Ya
4.	BEdit	Rich Siegel	Tidak	Tidak	Tidak	Ya
5.	BDV Notepad	Badevlad company	Ya	Tidak	Tidak	Tidak
6.	Bluefish	Bluefish Dev Team	Tidak	Ya	Ya	Ya
7.	Crimson Editor	Ingyu Kang	Ya	Tidak	Tidak	Tidak
8.	CRiSP	Paul D. Fox	Ya	Ya	Ya	Ya
9.	ConTEXT	Eden Kirin	Ya	Tidak	Tidak	Tidak
10.	Diakonos	Pistos	Ya	Ya	Ya	Ya
11.	ed	Ken Thompson	Ya	Ya	Ya	Ya
12.	EditPlus	ES-Computing	Ya	Tidak	Tidak	Tidak
13.	EditPad Lite	JustGreatSoftware	Ya	Ya	Tidak	Tidak
14.	EditPad Pro	JustGreatSoftware	Ya	Ya	Tidak	Tidak
15.	gedit	GNU Project	Tidak	Ya	Ya	Tidak
16.	GNU Emacs	Richard Stallman	Ya	Ya	Ya	Ya
17.	Kate	KDE Project	Tidak	Ya	Ya	Ya
18.	KWrite	KDE Project	Tidak	Ya	Ya	Ya
19.	jEdit	Slava Pestov	Ya	Ya	Ya	Ya
20.	JOE	Joseph Allen	Tidak	Ya	Ya	Ya
21.	Metapad	Alexander Davidson	Ya	Tidak	Tidak	Tidak
22.	Nano	GNU Project	Tidak	Ya	Ya	Ya
23.	NEdit	Mark Edel	Ya	Ya	Ya	Ya
24.	Notepad	Microsoft	Ya	Tidak	Tidak	Tidak
25.	Notepad++	Don Ho	Ya	Tidak	Tidak	Tidak
26.	Notepad2	Florian Balmer	Ya	Tidak	Tidak	Tidak
27.	Notetab	Eric Fookes, Fookes	Ya	Tidak	Tidak	Tidak

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

No	Nama Text Editor	Produsen	Lingkungan Sistem Operasi			
			Windows	Linux	Unix	Mac OS
		Software			k	
28.	nvi	Keith Bostic	Tidak	Ya	Ya	Ya
29.	Pico	University of Washington	Tidak	Ya	Ya	Ya
30.	PSPad	Jan Fiala	Ya	Tidak	Tidak	Tidak
31.	skEdit	Sean Kelly, skti	Tidak	Tidak	Tidak	Ya
32.	SubEthaEdit	TheCodingMonkeys	Tidak	Tidak	Tidak	Ya
33.	TED Notepad	Juraj Simlovic	Ya	Tidak	Tidak	Tidak
34.	TextEdit	Apple Computer	Tidak	Tidak	Tidak	Ya
35.	TextPad	Helios Software Solutions	Ya	Tidak	Tidak	Tidak
36.	TextMate	MacroMates	Tidak	Tidak	Tidak	Ya
37.	TextWrangler	Bare Bones Software	Tidak	Tidak	Tidak	Ya
38.	UltraEdit	IDM Computer Solutions	Ya	Tidak	Tidak	Tidak
39.	Vim	Bram Moolenaar	Ya	Ya	Ya	Ya
40.	XEmacs	Lucid Inc.	Ya	Ya	Ya	Ya
41.	Zeus	Xidicone P/L	Ya	Tidak	Tidak	Tidak

2.1. Bahasa Pemrograman

Bahasa pemrograman atau bahasa komputer merupakan teknik komunikasi standar untuk memerintah komputer. Bahasa pemrograman adalah suatu himpunan aturan sintaks dan semantik yang dipakai untuk mendefinisikan program komputer. Suatu bahasa memungkinkan seorang programmer secara persis menentukan data yang mana akan diolah oleh komputer, bagaimana data ini akan disimpan/diteruskan, dan langkah apa secara tepat yang akan diambil dalam berbagai situasi.

Komputer adalah mesin yang dapat melaksanakan seperangkat perintah dasar (*instruction set*). Maka agar komputer dapat melakukan sesuatu hal, kita harus memberinya perintah yang dapat ia laksanakan, yaitu dalam bentuk kumpulan perintah-perintah dasar tersebut.

Penerjemahan dilakukan oleh Program Komputer yang disebut Kompilator. Sintaks dari bahasa pemrograman lebih mudah dipahami oleh manusia daripada sintaks perintah dasar. Namun tentu saja komputer hanya dapat melaksanakan perintah dasar itu. Maka disinilah peran penting kompilator sebagai perantara antara bahasa pemrograman dengan perintah dasar.

Kegiatan membuat Program Komputer dengan menggunakan bahasa pemrograman disebut pemrograman komputer. Contoh bahasa pemrograman adalah bahasa FORTRAN, COBOL, BASIC, JAVA, C++, dan lain-lain.

Sejak dulu hingga sekarang, tentu kita mengetahui bahwa di dunia komputer terdapat beraneka ragam bahasa pemrograman. Karena begitu banyaknya jenis-jenis bahasa pemrograman, bahasa-bahasa tersebut juga dikelompokkan berdasarkan kriteria tertentu. Ada yang mengelompokkannya menjadi 3 level bahasa yaitu: *high level* (Seperti Pascal dan Basic), *middle level* (Seperti Bahasa C), dan *low level* (Seperti Bahasa Assembly). Ada juga yang mengelompokkan berdasarkan paradigma menjadi *procedural/functional programming*, *object oriented programming*, dan sebagainya.

Manusia dapat melakukan interaksi secara efektif dengan menggunakan media bahasa. Bahasa memungkinkan penyampaian gagasan dan pemikiran, tanpa itu komunikasi akan sulit terjadi. Dalam lingkungan pemrograman komputer, bahasa pemrograman bertindak sebagai sarana komunikasi antara manusia dan permasalahannya dengan komputer yang dipakai untuk membantu memperoleh pemecahan masalah. Bahasa pemrograman menjembatani antara pemikiran manusia yang sering tidak terstruktur dengan kepastian yang diperlukan oleh komputer untuk melakukan eksekusi.

Suatu solusi untuk suatu akan menjadi lebih mudah bila bahasa pemrograman yang dipergunakan lebih dekat dengan permasalahan tersebut. Oleh karena itu, bahasa harus memiliki konstruksi yang merefleksikan terminologi dan elemen yang dipergunakan dalam mendeskripsikan masalah dan independen dari komputer yang dipergunakan. Bahasa pemrograman seperti ini biasanya bahasa tingkat tinggi. Komputer digital, di sisi lain menerima dan memahami hanya bahasa tingkat rendah mereka sendiri, terdiri dari deretan nol dan satu yang sulit dipahami oleh manusia. Bahasa pemrograman berdasarkan tingkat ketergantungannya dengan mesin meliputi [UTD01]:

1. Bahasa mesin

Merupakan bentuk terendah dari bahasa komputer. Setiap instruksi dalam program direpresentasikan dengan kode numerik, yang secara fisik berupa deretan angka 0 dan 1. sekumpulan instruksi dalam bahasa mesin bisa dibentuk menjadi *microcode*, yaitu semacam prosedur dalam bahasa mesin.

2. Bahasa *assembly*

Merupakan bentuk simbolik dari bahasa mesin. Setiap kode operasi memiliki kode simbolik, misalnya ADD untuk penjumlahan (*addition*) dan MUL untuk perkalian (*multiplication*). Sekumpulan instruksi dalam bahasa *assembly* bisa dibentuk menjadi *makroinstruksi*. Pada bahasa *assembly* tersedia alat bantu untuk diagnostik atau *debug* yang tidak terdapat pada bahasa mesin. Contoh produk yang ada untuk pengembangan dan *debug* bahasa *assembly* di pasaran saat ini, misalnya Turbo Assembler dari Borland, Macro Assembler dari Microsoft, DEBUG yang tersedia pada DOS, dan Turbo Debugger. Instruksi dalam bahasa Assembler biasanya terdiri dari beberapa field, misalnya field operasi diikuti satu atau lebih operand.

3. Bahasa tingkat tinggi (*user oriented*)

Disebut tingkat tinggi karena lebih dekat dengan manusia. Memberikan fasilitas yang lebih banyak, kontrol program yang terstruktur, kalang (*nested*), *block*, dan prosedur. Contohnya: Pascal, BASIC.

4. Bahasa yang *problem oriented*

Memungkinkan penyelesaian untuk suatu masalah atau aplikasi yang spesifik. Contohnya: SQL (*Structured Query Language*) untuk aplikasi database, COGO untuk aplikasi teknik sipil. Bahasa yang *problem Oriented* kadang dimasukkan pula sebagai bahasa tingkat tinggi.

2.3. Perancangan Bahasa Pemrograman

2.3.1. Sumber Perancangan Bahasa Pemrograman

Gagasan untuk perancangan bahasa pemrograman bisa berasal dari bahasa alami (*natural language*), matematika, dan bahasa pemrograman yang sudah ada. Penjelasannya masing-masing sebagai berikut:

1. Konstruksi yang diturunkan dari bahasa alami berguna untuk kejelasan dan kemudahan pembacaan. Sebuah intruksi akan mengerjakan ‘mirip’ dengan arti instruksi itu. Hal ini memberikan kenyamanan untuk para pemrogram, khususnya yang belum berpengalaman. Di sisi lain bisa menimbulkan ambiguitas/ke-dwiarti-an yang tidak diinginkan dalam sebuah bahasa pemrograman. Bagaimanapun, bahasa alami bisa digunakan sebagai panduan untuk perancangan sintaks sebuah bahasa pemrograman.
2. Matematika telah banyak dipakai untuk aturan-aturan yang terdapat pada bahasa pemrograman, misalnya ekspresi aritmatika. Tetapi seorang pemrogram dan ahli matematika menggunakan metode dan memecahkan masalah yang berbeda. Meskipun matematika merupakan suatu sumber yang berguna, perlu kehati-hatian saat mengadopsi notasi matematika untuk suatu konsep.
3. Bahasa pemrograman yang sudah ada bisa menjadi sumber yang bagus untuk perancangan bahasa pemrograman. Tetapi perlu ketelitian saat menggunakannya, karena bahasa yang sudah ada itu mungkin mengandung kesalahan yang serius. Beberapa fasilitas yang diinginkan bisa dibatasi untuk meningkatkan kemudahan baca (*readability*) dan pemeriksaan kesalahan, dengan mengamati bagian mana dari bahasa yang jarang digunakan. Misalnya motivasi untuk tidak menggunakan instruksi GOTO datang dari pengamatan bahwa pemrogram yang baik tidak menggunakan GOTO untuk membuat struktur yang lebih mudah dipahami.

2.3.2. Tujuan Perancangan Bahasa Pemrograman

Perancangan sebuah bahasa pemrograman bertujuan untuk:

1. Komunikasi dengan manusia

Jika sebuah program sulit dimengerti oleh manusia, akan terjadi kesulitan juga untuk memeriksa dan melakukan pemeliharaan atau modifikasi. Masalah ini tidak bisa dipecahkan dengan sekadar menambahkan komentar

atau melalui dokumentasi tersendiri. Pemrogram tidak suka menambah komentar yang berlebihan, sementara dokumentasi program sering tidak lengkap dan sudah kadaluwarsa. Sebuah program yang mudah dibaca atau dimengerti adalah sebuah dokumentasi yang baik. Sintaks sebuah bahasa pemrograman harus merefleksikan semantiknya. Keterbatasan kemampuan berpikir manusia menyulitkan untuk memahami suatu struktur yang kompleks, sementara kompilator tidak. Misalkan algoritma manipulasi dengan *stack*. Mungkin suatu *statement* bermakna ambigu bagi manusia, sementara sudah cukup jelas bagi kompilator, misalnya instruksi aritmatika berikut:

$$a/b/c$$

bisa berarti

a dibagi dengan b, baru hasilnya dibagi dengan c

atau

a dibagi dengan hasil pembagian b dan c

2. Pencegahan dan deteksi kesalahan

Sebuah bahasa pemrograman yang baik perlu mengidentifikasi *error* yang mungkin terjadi. Sehingga mempermudah deteksi kesalahan yang menghilangkannya.

3. *Usability*

Bahasa pemrograman harus mudah dipelajari dan diingat. Sekali seorang pemrogram familiar dengan bahasa itu, dia tidak harus melihat manual terus menerus. *Usability* berkaitan dengan aspek kenyamanan seorang pemrogram menggunakan bahasa.

4. Efektifitas pemrograman

Efektifitas di sini berkaitan dengan pemrograman sebagai bagian dari rekayasa perangkat lunak. Hal yang menjadi perhatian besar pada rekayasa perangkat lunak adalah bagaimana mencatat keputusan yang dibuat selama pengembangan program. Program sendiri adalah tempat terbaik untuk mencatat keputusan tersebut. Bahasa pemrograman harus memfasilitasi suatu *statement* yang jelas dari keinginan pemrogram. Bilamana mungkin, bahasa harus memungkinkan pemrogram untuk menyatakan keinginan mereka dan

membiarkan kompilator melakukan implementasinya. Suatu bahasa yang jelas dan mudah dibaca akan mencegah pemrogram yang ‘cerdas’ untuk melakukan ‘tricky’. Salah satu contoh rancangan presedensi operator yang buruk adalah di dalam bahasa C, yang memiliki selusin tingkat prioritas.

5. *Compilability*

Pengembangan sebuah kompilator merupakan pekerjaan yang tidak sederhana. Sebuah bahasa yang terlalu kompleks akan menyulitkan pembuatan kompilator untuk bahasa tersebut. Kompleksitas tersebut bisa muncul baik pada tahapan analisis maupun sintesis. Kompleksitas biasanya muncul pada bahasa di mana koma dan *parenthesis* (kurung buka atau kurung tutup) digunakan pada banyak aturan yang berbeda. Sebagai contoh, *parenthesis* bisa dipakai pada:

- a. Pengelompokkan sub-ekspresi
- b. Membatasi argumen pada pemanggilan prosedur
- c. Membatasi parameter pada definisi prosedur
- d. Menentukan indeks pada *array*
- e. Ekspresi logika

Dalam kasus tersebut diperlukan *look-ahead parser* yang mampu melihat dua atau tiga token ke depan, sehingga memperlambat kompilasi dan menambah ukuran tabel untuk parsing. *Feature* bahasa yang kompleks tapi jarang dibutuhkan lebih baik tidak dimasukkan. Salah satu contoh skema yang sederhana tetapi *powerful* adalah metode *parsing parameter* dengan *call by value* dan *call by reference*. Sebisa mungkin rancangan bahasa dibuat selengkap mungkin sebelum masuk ke tahapan implementasi kompilator.

6. Efisiensi

Efisiensi merupakan bahasan yang penting dalam sejarah pengembangan bahasa pemrograman. Dalam hal ini harus diingat pula bahwa mesin akan lebih murah, sementara pemrogram akan semakin mahal. Kita tidak bisa memaksakan efisiensi 10 persen dengan usaha yang dikeluarkan cukup keras, karena hasilnya tidak sebanding dengan ‘investasi’ yang dilakukan. Perbedaan efisiensi 10 atau 20 persen bisa ditoleransi. Tetapi tidak untuk perbedaan dalam kelipatan 2 atau 10 karena alasan-alasan berikut:

- a. Bagaimanapun dan murah nya perangkat keras, akan lebih murah dan cepat bilamana menjalankan sebuah program yang efisien.
- b. Kecepatan dan harga dari *peripheral* tidak meningkat secepat CPU.
- c. Tidak seharusnya *user* dikorbankan dengan memaksa mereka mengeluarkan uang untuk peningkatan perangkat keras.

Salah satu hal yang penting adalah menentukan penyebab utama dari inefisiensi. Penyebab terbesar biasanya adalah ketidakcocokan antara perangkat keras dengan bahasa. Mencoba untuk membuat bahasa terlalu ‘dekat’ dengan perangkat keras juga akan memberikan hasil yang buruk. Optimasi suatu kompilator bukanlah jawaban utama. Suatu optimasi kompilator menimbulkan kesulitan dan memerlukan banyak waktu untuk dibuat. Harus dilakukan secara hati-hati supaya semantik dari bahasa tidak berubah karena proses optimasi. *Optimizer* umumnya juga berjalan lebih lambat daripada kompilator sederhana. Efisiensi bukanlah hanya permasalahan ukuran kecepatan, karena akses I/O dan pembagian memori termasuk juga dalam efisiensi. Peningkatan yang signifikan dalam efisiensi bisa didapat dengan membuat bahasa lebih sederhana sehingga mudah untuk menghasilkan kode yang efisien.

7. *Machine Independent*

Sebuah bahasa dikatakan *machine independent* jika dan hanya jika sebuah program yang telah sukses dikompilasi dan dieksekusi pada suatu mesin, saat dipindahkan ke mesin lainnya akan berjalan dengan input dan output yang tepat sama. Perlu diperhatikan *machine independence* tidak diharuskan untuk semua bahasa pemrograman. Misalnya bahasa yang ditujukan untuk penulisan sistem operasi bisa memiliki ketergantungan tertentu pada mesin. Beberapa masalah yang berkaitan dengan independensi suatu bahasa misalnya:

- a. Aritmatika *floating point*
- b. Kumpulan (set) karakter yang ada
- c. Karakter kontrol

8. Kesederhanaan (*simplicity*)

Kebanyakan pemrogram menginginkan bahasa yang sederhana. Tetapi kesederhanaan bisa pula berarti kekurangan di sisi lain. Bahasa BASIC, contohnya, merupakan bahasa yang sederhana, tetapi tidak dirancang secara baik. *Simplicity* tidak bisa dicapai dengan keterbatasan struktur yang akan menghasilkan *chaos*. *Simplicity* juga tidak bisa dicapai dengan generalitas yang tidak terbatas, karena akan menghasilkan bahasa yang sangat sulit untuk diimplementasikan secara lengkap. Kesederhanaan bisa dicapai melalui:

- a. Pembatasan-pembatasan tujuan
- b. Perhatian pada keterbacaan
- c. Pendefinisian yang baik
- d. Konsep yang sederhana

9. *Uniformity*

Uniformity bisa didefinisikan sebagai mengerjakan hal yang sama dengan cara yang sama. Bila diadopsi sebagai prinsip bahasa pemrograman, ini bisa menolong mengurangi hal-hal yang harus diingat pemrogram, karena akan lebih mudah memahami bagaimana suatu *feature* akan dilakukan. Seperti aspek-aspek yang lain, *uniformity* bisa menjadi berguna atau tidak tergantung dari bagaimana penerapannya. Suatu contoh penerapan *uniformity* yang berguna misalkan pemakaian ekspresi pada ALGOL, dimanapun dibutuhkan suatu nilai aritmatika bisa digunakan ekspresi apa pun. Contoh penerapan *uniformity* yang tidak berguna misalnya gagasan mengenai ekspresi dan *statement* pada ALGOL. Suatu bahasan dalam kasus *uniformity* adalah bagaimana menghilangkan kasus khusus. Kasus-kasus khusus seperti ini memiliki masalah:

- a. Membuat bahasa menjadi lebih kompleks
- b. Menuju pada implementasi yang berbeda, beberapa akan mengimplementasikannya dengan tidak tepat, yang lain mungkin akan menghilangkannya.
- c. Menjadi lebih sulit dihilangkan meskipun terdapat kekurangannya.

10. *Orthogonality*

Gagasan mendasar dari *orthogonality* adalah setiap fungsi bekerja tanpa perlu mengetahui struktur dari yang lain. Misalkan operasi aritmatika

menggabungkan dua nilai, tidak memperdulikan bagaimana nilai tersebut didapatkan. Alternatif untuk *orthogonality* adalah *diagonality*.

11. Generalisasi dan spesialisasi

Pemikiran mendasar dari generalisasi adalah: “jika kita dibolehkan melakukan ini, maka lakukanlah hal lain dengan cara serupa”. Dalam cara serupa dengan *orthogonality*, merupakan pelengkap dari *simplicity*. Jika digunakan berlebihan, bisa menghasilkan pemakaian yang jarang, atau *feature* yang mengandung kesalahan yang sulit diimplementasikan. Pada beberapa bahasa terdapat fungsi *built-in* yang menerima beberapa pun jumlah argumen, di mana jarang *user* menggunakannya.

2.4. Model Kompilator

Pengembangan kompilator untuk sebuah bahasa merupakan pekerjaan yang kompleks. Kompleksitas kompilator bisa dikurangi bila perancangan bahasa pemrograman mempertimbangkan bermacam-macam faktor perancangan. Karena kita berhubungan dengan bahasa tingkat tinggi, bagaimanapun suatu model dasar dari kompilator dapat diformulasikan. Sebuah kompilator umumnya memiliki dua tugas pokok:

1. Fungsi Analisis

Fungsi analisis biasa disebut sebagai *Front End*. Tugasnya melakukan dekomposisi program sumber menjadi bagian-bagian dasarnya.

2. Fungsi Sintesis

Fungsi sintesis biasa disebut sebagai *Back End*. Tugasnya melakukan pembangkitan dan optimasi program objek.

Gambar 2.1 Model Kompilator [UTD01]

Keterangan gambar:

: Proses yang dapat dilakukan oleh PuniEdit

- a. *Scanner* : Memecah besaran sumber menjadi besaran leksik
- b. *Parser* : Memeriksa kebenaran dan urutan kemunculan token.
- c. Analisis Semantik : Melakukan analisa semantik, biasanya dalam realisasi akan digabungkan dengan *intermediate code generator* (bagian yang berfungsi membangkitkan kode antara)
- d. *Code Generator* : Membangkitkan kode objek
- e. *Code Optimizer* : Memperkecil hasil dan mempercepat proses.
- f. Tabel Simbol : Menyimpan semua informasi yang berhubungan dengan proses kompilasi.

2.5. Analisis Leksikal

2.5.1. Tugas *Scanner*

Scanner bertugas melakukan analisis leksikal, yaitu mengidentifikasi semua besaran yang membangun suatu bahasa pada suatu program sumber. *Scanner* adalah bagian dari kompilator yang menerima input berupa *stream* karakter kemudian memilah program sumber menjadi satuan leksik yang disebut dengan *token*. Token ini akan menjadi input bagi *parser*.

Tugas *Scanner* adalah sebagai berikut:

1. Melakukan pembacaan kode sumber dengan karakter demi karakter
2. Mengenali besaran leksik
3. Mentransformasi menjadi sebuah *token* dan menentukan jenis *token*-nya
4. Mengirimkan *token*
5. Membuang atau mengabaikan *blank* dan komentar dalam program
6. Menangani kesalahan
7. Menangani tabel simbol

Scanner bekerja berdasarkan mesin *Finite State Automata* yang ada pada Bahasa Regular. Untuk membantu mengkonstruksi *Scanner* dapat mempergunakan Diagram Keadaan.

2.5.2. Besaran Leksik

Besaran Pembangunan bahasa/leksik meliputi:

1. *Identifier*

Bisa berupa *keywords* atau nama. *Keywords* atau kata kunci yang sudah didefinisikan oleh suatu bahasa seperti BEGIN, END, IF, ELSE di dalam Pascal. Nama dideklarasikan sendiri oleh pemakai, seperti nama sebuah variabel misalnya. Contoh bila pada suatu program Pascal terdapat deklarasi:

```
VAR
    Nomor : INTEGER;
    Suhu : REAL;
```

Maka Nomor dan Suhu akan dikenali sebagai besaran leksik berupa nama variabel yang terdapat pada program tersebut. Sedangkan VAR, INTEGER, dan REAL merupakan *Keyword* pada program tersebut. Nama bisa berupa nama program, procedure, var, type, constant.

2. Nilai konstanta

Adalah suatu konstanta yang terdapat pada program. Bisa berupa konstanta integer, real, boolean, character, string, dan sebagainya. Misalkan saja dalam Pascal pada suatu program terdapat *statement*:

```
N := R + 5 * 10
kata := kata1 + 'makan'
A := 0.333
Selesai := TRUE
```

Maka 5, 10, 'makan', 0.333, TRUE, termasuk besaran leksik bernilai konstanta yang terdapat pada program sumber tersebut.

3. Operator dan *delimiter*

Operator misalnya operator aritmatika (+, -, *, /,), operator logika (<, =, >).

Delimiter berguna sebagai pemisah atau pembatas, misalnya: (), ;, :, (kurung buka/tutup, koma, titik, titik koma, titik dua), *white-space*. *White-space* adalah pemisah yang diabaikan di program, seperti spasi, karakter *enter* (*Carriage Return*), ganti baris (*Line Feed*), akhir file (*End of File*). Contoh beberapa *delimiter*:

^	,	.
;	(	
..	)	]
[	:	

Misalkan terdapat sebuah *source program*:

```
PROGRAM Coba;  
VAR A : INTEGER  
BEGIN  
    A := A+2;  
END.
```

Pada contoh di atas, besaran leksik (*token*)-nya adalah simbol yang bernilai 'PROGRAM', 'Coba', 'VAR', 'A', 'INTEGER', ':', '+', '2', '.', ';', ':=', 'BEGIN', 'END'.

Misalkan sebuah bahasa memiliki himpunan simbol terminal atau *token*:

'<', '>', '=', '<=', '>=', '<>'

atau bisa dibaca sebagai *token-token*:

$t_L, t_G, t_E, t_{GE}, t_{NE}$.

($G=greater$, $L=less$, $E=equal$, $N=not$). Bahasa tersebut juga mendukung penulisan komentar yang diawali dengan '{' dan diakhiri dengan '}'. Kita lihat Diagram Keadaan untuk kumpulan *token* di atas pada gambar 2.2.

Dalam Pascal, komentar tidak diperlukan lagi pada proses selanjutnya, karena itu komentar tidak dimasukan sebagai *token*. Setiap *scanner* menemukan awal komentar, *scanner* hanya mengambil simbol yang didapat tanpa disimpan ke dalam *token*. Setelah ditemukan akhir dari komentar, *state* dikembalikan ke *state* awal (serupa dengan *white space*).

Scanner biasanya diimplementasikan sebagai sebuah prosedur yang dipanggil oleh *Parser*. Kita bisa membuat prosedur *scan* sederhana yang akan membaca sebuah file *input* dan memberikan *token* hasilnya. Misal didefinisikan dulu *Procedure GetChar* yang berguna mengambil sebuah karakter dari file *input*.

Gambar 2.2 Diagram Keadaan [UTD01]

2.6. Pemrosesan Teks

2.6.1. Susunan Teks

Mengakses data yang tersusun secara beruntun ini akan sering kita jumpai dalam pemrograman. Contoh data seperti itu adalah larik, arsip beruntun (*sequential file*), arsip teks (*teks file*) dan senarai (*linked list*).

Arsip teks untuk selanjutnya kita sebut teks adalah arsip yang terdiri atas deretan karakter. Untaian karakter itu dapat membentuk sebuah “kata”. Yang dimaksud dengan kata adalah kelompok karakter yang dipisahkan dengan kelompok lain dengan satu atau lebih spasi.

Pada implementasinya di dalam teks ditambahkan akhir baris (*end-of-line*) yang membedakan sebuah baris teks dengan baris lainnya (*end-of-line* terdiri dari karakter *line-feed*). Contoh teks adalah program sumber (*source program*) Pascal (yaitu arsip *.PAS) atau C (arsip *.C), arsip README.TXT pada beberapa program aplikasi tertentu, arsip AUTOEXEC.BAT pada sistem operasi DOS, dan lain-lain. Teks dapat dibuat dengan pengolah kata teks, seperti *Notepad*, *SideKick*, editor kompilator bahasa pemrograman seperti Turbo Pascal atau Turbo C, atau dengan pengolah kata *Microsoft Word* asalkan disimpan sebagai (*save as*) sebagai arsip teks (*Text Only*).

Halo, apa kabar? Saya di sini sedang belajar algoritma dan pemrograman; belajar algoritma dan pemrograman yang efektif adalah dengan banyak membuat latihan.

Gambar 2.3 Contoh sebuah teks yang terdiri atas 5 buah baris. Akhir teks adalah karakter titik (‘.’) [MUN99].

Pemrosesan teks cukup banyak ditemui di dalam aplikasi pemrograman. Misalnya program pengolah kata (*text editor*) adalah program yang mengolah teks. Pemrosesan teks antara lain proses mencari (*search*) sebuah karakter atau string, menghitung jumlah karakter atau jumlah kata, menempatkan *pointer* pembacaan ke karakter/kata tertentu, dan sebagainya. Aplikasi lain yang memproses teks adalah kompilator bahasa pemrograman. Tugas pertama kompilator adalah membaca program sumber untuk mengambil simbol-simbol (*keyword*, peubah, tetapan, dan sebagainya). Komponen kompilator yang melakukan aktivitas ini dinamakan program *scanner*.

2.6.2. Tanda Akhir Teks

Pemrosesan teks yang utama adalah proses pembacaan. Karena teks berisi runtunan karakter, maka pembacaan teks adalah membaca karakter demi karakter secara beruntun, mulai dari awal teks sampai akhir teks. Akhir teks ditandai dengan

sebuah karakter khusus, yang dibedakan dengan karakter-karakter lainnya. Bila pembacaan teks tertentu dengan karakter khusus tersebut, maka proses pembacaan teks dihentikan.

Di dalam Penelitian ini, kita mendefinisikan karakter titik (‘.’) sebagai akhir teks (tetapi, ini bukan satu-satunya karakter akhir teks yang dapat digunakan. Anda juga dapat menggunakan karakter yang lain seperti ‘#’, ‘*’, ‘\$’, dan sebagainya, asalkan di dalam teks itu sendiri karakter tersebut tidak pernah digunakan). Karakter titik ditaruh sesudah karakter terakhir. Kita mengartikan karakter titik bukan bagian dari elemen teks. Sekali lagi, karakter titik hanya sebagai penanda akhir arsip. Sembarang teks yang kita susun harus diakhiri dengan titik agar algoritma yang kita susun berlaku. Gambar 2.4 memperlihatkan sebuah teks yang terdiri atas 11 karakter, yaitu karakter ‘l’, ‘a’, ‘y’, ‘a’, ‘n’, ‘a’, ‘n’, ‘l’, ‘o’ dan ‘8’ (karakter titik tidak dihitung sebagai unsur teks).

Layanan 108

Gambar 2.4 Teks yang terdiri atas 11 karakter. Akhir teks adalah ‘.’ [MUN99]

2.6.3. Definisi Teks Kosong

Pemrosesan teks selalu mempertimbangkan keadaan awal teks, yaitu kosong atau tidak kosong. Sebuah teks mungkin saja kosong. Pertama-tama kita harus mendefinisikan makna kosong itu sendiri. Sebuah arsip yang hanya berisi sebuah spasi tidak dapat disebut kosong. Karena kita telah membuat perjanjian bahwa sembarang teks harus diakhiri dengan karakter titik, maka teks kosong adalah teks yang hanya berisi karakter kosong, jumlah unsurnya adalah 0. Gambar 2.5 memperlihatkan sebuah teks kosong.

.

Gambar 2.5. Sebuah teks kosong hanya berisi karakter ‘.’ [MUN99].

2.6.4. Pembacaan Teks

Pembacaan teks dimulai dari awal teks (dari “kiri” ke “kanan”). Kita mengandaikan ada sebuah *pointer* (dilambangkan dengan “↑”) yang menunjuk ke karakter yang akan dibaca. Setiap kali karakter yang ditunjuk oleh *pointer* selesai dibaca, *pointer* berpindah secara otomatis ke karakter berikutnya. Karakter yang sedang ditunjuk oleh *pointer* dinyatakan di dalam peubah C. Gambar 2.6 memperlihatkan *pointer* menunjuk karakter ‘M’. jika karakter ini dibaca, maka C berisi ‘M’. Karakter ‘K’ dan karakter ‘R’ sudah dibaca sebelumnya, sedangkan karakter S, 9, \$ dan seterusnya belum dibaca. Pembacaan teks hanya dilakukan selama $C \neq \text{'.'}$. Jika $C = \text{'.'}$, dikatakan akhir pita telah dicapai dan pembacaan pita dihentikan.

KRMS9\$SATE.

↑ *pointer*

Gambar 2.6 Karakter yang sedang ditunjuk oleh *Pointer* baca (dilambangkan dengan ↑). ‘M’ adalah karakter yang ditunjuk oleh *pointer* baca. Peubah C menyimpan karakter yang ditunjuk oleh *pointer* baca, dalam hal ini, C berisi ‘M’. [MUN99]

2.7. Rekayasa Perangkat Lunak

Perangkat lunak merupakan suatu program yang berisi sejumlah instruksi yang dibuat untuk melakukan sejumlah proses dengan tujuan sebagai alat pengendali kegiatan sistem komputer atau untuk menyelesaikan aplikasi tertentu [PRE02].

Dalam membangun suatu perangkat lunak selalu timbul berbagai macam kesulitan. Kesulitan yang terdapat dalam pembangunan perangkat lunak dapat diatasi dengan cara mengenali permasalahan yang ada dan penyebab timbulnya permasalahan tersebut. Tapi solusi ini harus dapat memberikan bantuan yang berguna bagi pembangunan perangkat lunak, meningkatkan kualitas perangkat lunak, dan akhirnya menerapkan perangkat lunak pada perangkat keras.

2.7.1. Definisi

Rekayasa perangkat lunak merupakan pembuatan dan penggunaan prinsip-prinsip rekayasa dalam hal memperoleh atau menghasilkan perangkat lunak secara ekonomis, yang dapat diandalkan dan bekerja tepat guna pada suatu perangkat keras [PRE02].

2.7.2. Pendekatan Pembangunan Perangkat Lunak

Beberapa metodologi pembangunan perangkat lunak dikembangkan orang. Masing-masing metodologi memiliki kelebihan dan kekurangan. Walaupun demikian setiap pendekatan yang ada berusaha agar dapat menghasilkan suatu perangkat lunak yang baik dan sesuai dengan kebutuhan pengguna.

Classic Life Cycle atau model sekuensial linear atau biasa juga disebut dengan model *waterfall*. Siklus ini bersifat sistematis, dengan pendekatan sekuensial untuk membangun perangkat lunak, dimulai pada tingkat sistem berkembang sampai dengan pengetesan, dan perbaikan kesalahan.

Model *Classic Life Cycle* adalah paradigma rekayasa perangkat lunak yang paling luas dipakai dan paling tua. Paradigma model ini memiliki tempat yang terbatas namun penting di dalam kerja rekayasa perangkat lunak. Paradigma itu memberikan *template* dimana metode analisis, desain, pengkodean, pengujian dan pemeliharaan bisa dilakukan. Siklus metode *Classic Life Cycle* tetap menjadi model bagi rekayasa perangkat yang paling luas dipakai. Sekalipun memiliki kelemahan, secara signifikan lebih baik daripada pendekatan yang sifatnya sembarang kepada pengembangan perangkat lunak.

Gambar 2.7 Model Classic Life Cycle atau Model Waterfall [PRE02]

2.8. Perancangan Antar Muka

Antar muka suatu perangkat lunak menggambarkan bagaimana cara sebuah program akan dilihat dan digunakan oleh pengguna. Perancangan sebuah antar muka termasuk didalamnya mendefinisikan layar, informasi keluaran, dan struktur menu.

Hal-hal yang perlu diperhatikan dalam merancang *user interface* yaitu:

1. *Real word metaphors*

Berdasarkan pada pandangan pengguna dari sistem tentang sesuatu yang nyata di dunia daripada sesuatu yang terdapat dalam sistem komputer. Misalnya, layar masukan harus dirancang untuk menggambarkan bentuk/format dokumen yang digunakan orang dalam kehidupan nyata daripada menggambarkan struktur dari basis data atau file di dalam suatu komputer.

2. *Seleksi dari pada mengingat*

Pengguna tidak harus mengingat nama-nama dari perintah, file atau yang serupa. Hal ini biasanya berkaitan dengan penggunaan menu-menu. Pemakai tidak harus memberikan informasi dengan segera ke komputer dan tidak perlu hafal dengan sistem. Misalnya, suatu modul untuk mempertahankan keutuhannya yang memberikan pemakai

satu pilihan untuk menambah, mengubah dan menghapus, pemakai tidak harus ditanya dari mana suatu file yang akan digunakan tetapi seharusnya diberikan suatu daftar dari file yang valid yang dapat dipilih oleh pengguna.

3. *Konsisten*

Semua layar dari menu sedapat mungkin harus terlihat dan bekerja serupa. Bila menggunakan *function key*, *key* yang sama harus berarti sama untuk setiap menu. Pada setiap menu, perintah yang sama harus berada pada tempat yang sama.

4. Kontrol user

Pengguna harus dapat memilih apa yang dia kerjakan dalam urutan dimana dia akan lakukan. Misalnya, suatu sistem tidak boleh mensyaratkan bahwa semua record yang baru ditambahkan sebelum record yang ada dapat diubah. Hal yang sama, apabila operasi akan berlangsung lama (seperti mengurutkan dan mengindeks suatu file yang besar), pengguna harus mempunyai pilihan untuk memulai suatu operasi daripada harus dipaksa mengambil langkah operasi daripada harus dipaksa mengambil langkah operasi yang telah ditentukan.

5. Umpan balik

Pengguna harus terus diberikan informasi tentang apa yang dikerjakan oleh sistem. Misalnya, layar harus mempunyai judul sehingga pengguna mengetahui bahwa dia memilih layar yang benar. Demikian pula, apabila suatu operasi yang panjang seperti pengurutan sedang berjalan, layar harus menyangkan suatu pesan sehingga pengguna mengetahui bahwa komputer tidak *hang*. Demikian pula, apabila pengguna memilih suatu record untuk dihapus, record harus ditayangkan pada layar untuk konfirmasi bahwa record itulah yang diinginkan untuk dihapus.

6. Pemaaf

Sistem tidak boleh bertindak sebelum pengguna harus diberi kesempatan untuk memperbaikinya. Misalnya, setelah pengguna memilih suatu record untuk dihapus, sistem harus ditanya kembali untuk konfirmasi, sebelum perintah untuk menghapus record benar-benar dilaksanakan.

7. Merasakan stabilitas

Pengguna tidak harus memperhatikan perubahan-perubahan sistem kecuali jika perubahan-perubahan tersebut berdampak pada pekerjaan pengguna. Misalnya,

menambahkan transaksi baru, atau satuan baru tidak boleh berdampak pada struktur dari menu yang ada dan secara absolute tidak boleh mengubah arti dari masukan pengguna untuk melakukan tugas lain.

3. Deskripsi Global Perangkat Lunak dan Deskripsi Rinci Kebutuhan

3.1. Deskripsi Global Perangkat Lunak

3.1.1. Perspektif Produk

Pada bagian ini akan dijelaskan bagaimana perangkat lunak PuniEdit ini menjalankan fungsi-fungsi yang ada. Perangkat lunak PuniEdit menerima input kode sumber dari pengguna berupa buka kode sumber, buat kode sumber, edit kode sumber, dan pengelolaan token, kemudian kode sumber akan diproses dengan fungsi *scan* kode sumber untuk mengetahui bahasa pemrograman yang dipakai serta menampilkan jenis token pada layar monitor atau dicetak melalui printer.

Berikut ini adalah gambar diagram blok yang akan menunjukkan gambaran umum dari perangkat lunak PuniEdit.

3.1.2. Fungsi Produk

Fungsi yang terdapat pada perangkat lunak PuniEdit adalah:

1. Fungsi Pengelolaan Kode sumber

Fungsi ini berguna untuk mengelola kode sumber. Secara khusus fungsi ini dapat melakukan:

1.1. Buka Kode sumber

Subfungsi ini merupakan input kode sumber berupa file.

1.2. Edit Kode sumber

Subfungsi ini merupakan fungsi untuk melakukan perubahan terhadap kode sumber, setiap perubahan akan selalu dilakukan scan kode sumber.

1.3. Simpan Kode sumber

Subfungsi ini akan melakukan pemeriksaan bahasa pemrograman yang sedang digunakan dan melakukan penyimpanan kode sumber dalam bentuk file, penyimpanan kode sumber ini ada dua yaitu penyimpanan dengan nama file yang sama (*save*) dan penyimpanan dengan nama file yang berbeda (*save as*).

1.4. Buat Kode sumber

Subfungsi ini akan melakukan pembuatan kode sumber baru, pada subfungsi ini pengguna diharuskan untuk memilih jenis bahasa pemrograman yang akan digunakan untuk mengenali jenis token.

2. Fungsi Scan Kode Sumber

Fungsi ini berguna untuk mengenali jenis bahasa pemrograman dan kemunculan jenis token. Dan fungsi ini dapat mengidentifikasi tipe file dari bahasa pemrograman.

3. Fungsi Pengelolaan Token

Fungsi ini berguna untuk mengelola data token yaitu menyimpan data token dalam bentuk file bertipe *.hgl sehingga data tersebut dapat dimanipulasi sesuai dengan kebutuhan, Secara khusus modul ini berfungsi untuk:

3.1. Input data token

3.2. Ubah data token

3.3. Hapus data token.

4. Fungsi Pencetakan Kode sumber

Fungsi ini berguna untuk mencetak kode sumber yang telah terformat ke media kertas melalui printer.

3.1.3. Karakteristik Pengguna

Programmer dengan keterampilan memahami bahasa pemrograman dan keterampilan dalam menjalankan perangkat lunak PuniEdit.

Struktur Menu

Gambar 3.2 Struktur Menu

3.2. Kebutuhan Fungsional

Pada bagian ini akan dijelaskan mengenai informasi yang diinput kemudian diproses dan menghasilkan informasi keluaran juga berdasarkan fungsi-fungsi yang ada pada perangkat lunak.

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

Gambar 3.3 Diagram Konteks Perangkat Lunak PuniEdit

Gambar 3.4 DFD Level 1 Perangkat Lunak PuniEdit

a. Aliran Informasi

Berikut ini adalah proses-proses dari setiap modul.

1. Pengelolaan kode sumber

Proses terdiri dari empat bagian yaitu proses buka kode sumber, edit kode sumber, buat kode sumber, dan simpan kode sumber.

Proses buka kode sumber memenuhi spesifikasi sebagai berikut:

- a) Menerima data berupa kode sumber dalam bentuk file dari pengguna.
- b) Memberikan tipe file pada scan kode sumber.
- c) Memberikan kode sumber pada scan kode sumber.

Proses edit kode sumber memenuhi spesifikasi sebagai berikut:

- a) Menerima kode sumber dari pengguna.
- b) Memberikan kode sumber pada scan kode sumber.
- c) Mendapatkan kode sumber, jenis token, dan jenis bahasa pemrograman.
- d) Mendapatkan kode sumber yang dimaksud.
- e) Mengubah kode sumber tersebut.
- f) Menyimpan kode sumber yang telah diubah.

Proses buat kode sumber memenuhi spesifikasi sebagai berikut:

- a) Menentukan jenis bahasa pemrograman yang digunakan.
- b) Menerima kode sumber dari pengguna.
- c) Mengirimkan kode sumber pada scan kode sumber.
- d) Mengirimkan jenis bahasa pemrograman pada scan kode sumber.
- e) Mendapatkan kode sumber, jenis token, dan jenis bahasa pemrograman.
- f) Menyimpan kode sumber yang telah dibuat.

Proses simpan kode sumber memenuhi spesifikasi sebagai berikut:

- a) Menerima kode sumber dari pengguna.
- b) Mengirimkan jenis bahasa pemrograman pada scan kode sumber.
- c) Mendapatkan tipe file dari bahasa pemrograman yang dimaksud.
- d) Menyimpan kode sumber tersebut dalam bentuk file.

Gambar 3.5 DFD Level 2 Fungsi Pengelolaan Kode sumber

2. Scan kode sumber

Proses scan kode sumber tidak mempunyai turunan, hanya memenuhi spesifikasi sebagai berikut:

- a) Menerima data tipe file dari proses buka kode sumber.
- b) Menerima data kode sumber dari proses buka kode sumber, edit kode sumber, buat kode sumber dan simpan kode sumber.
- c) Memberikan informasi jenis bahasa pemrograman, jenis token, dan tipe file dari bahasa pemrograman pada pengguna.

3. Pengelolaan Token

Proses terdiri dari tiga bagian yaitu proses input data token, ubah data token, dan hapus data token.

Proses input data token memenuhi spesifikasi sebagai berikut:

- a) Menerima data berupa nama file, dan data token dari pengguna.
- b) Memberikan konfirmasi input data token pada pengguna.
- c) Menyimpan data token dengan nama file yang bersangkutan.

Proses ubah data token memenuhi spesifikasi sebagai berikut:

- a) Menerima data nama file dari pengguna.
- b) Mencari data token
- c) Mendapatkan data token tersebut.
- d) Mengubah data token tersebut.
- e) Memberikan konfirmasi ubah data token pada pengguna.
- f) Menyimpan data token yang telah diubah.

Proses hapus data token memenuhi spesifikasi sebagai berikut:

- a) Menerima data nama file dari pengguna.
- b) Memberikan konfirmasi hapus data token.
- c) Menghapus nama file tersebut.

Gambar 3.6 DFD Level 2 Pengelolaan Token

4. Pencetakan kode sumber

Proses pencetakan kode sumber tidak mempunyai turunan, hanya memenuhi spesifikasi sebagai berikut:

- a) Menerima kode sumber yang telah terformat.
- b) Menampilkan kode sumber yang telah terformat tersebut.

b. Kamus Data

Data_token = 1{Legal_Character}n

Delimiter = [! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / | : | ; | < | = | > | ? | @ | [| \ |] | ^ | ` | { | | | } | ~ |]

Jenis_bahasa_pemrograman = 1{Legal_Character}n

Jenis_token = 1{Legal_character}n + 1{Delimiter}n

Kode_sumber = 1{Legal_character}n

Kode_sumber_yang telah terformat = 1{Legal_character}n

Legal_character = [A-Z | a-z | 0-9 | - | |]

Nama_file = 1{Legal_character}20

Tipe_file = * + . + 1{Legal_Character}6

3.3. Deskripsi Kebutuhan Non Fungsional

Tujuan deskripsi kebutuhan non fungsional adalah untuk menspesifikasikan ukuran kuantitatif yang harus dipenuhi oleh perangkat lunak. Memori yang digunakan pada sistem komputer pendukung perangkat lunak terdiri dari memori primer dan sekunder. Berikut adalah spesifikasi memori yang digunakan perangkat lunak.

Tabel 3.1 Batasan Memori

Jenis Memori	Spesifikasi	
Primer	Maksimum	512 Mbyte
	Minimum	128 MByte
Sekunder	Maksimum	80 Gbyte
	Minimum	10 Gbyte

Agar saat eksekusi program aplikasi semua unit program yang digunakan pada perangkat lunak tersebut akan di load ke memori utama. Sehingga untuk

mengoperasikan perangkat lunak yang mempunyai cukup banyak modul program memerlukan kapasitas memori utama yang cukup besar pula. Maka atas dasar hal tersebut, penulis mencantumkan kebutuhan memori primer sebesar itu.

Alasan kebutuhan tempat penyimpanan sekunder dengan kapasitas sebesar itu adalah untuk keperluan penyimpanan data dan kebutuhan perangkat lunak pendukung lainnya.

4. Perancangan Perangkat Lunak

4.1. Rancangan Lingkungan Implementasi

Pada rancangan lingkungan implementasi terdapat beberapa hal yang harus ditetapkan, dengan rincian sebagai berikut:

1. *Operating System*
Microsoft Windows XP Professional
2. *Development Tools*
Borland Delphi 7.0
3. Bahasa Pemrograman yang Dipakai
Object Pascal

4.2. Dekomposisi Fungsional Modul

Berikut adalah dekomposisi fungsional modul pada perangkat lunak PuniEdit yang akan dijelaskan pada tabel dibawah ini, yaitu:

Tabel 4.1 Dekomposisi Fungsional Modul

No	Modul	Proses
1.	Pengelolaan kode sumber	a. Memasukan kode sumber b. Memanipulasi kode sumber
2.	Scan kode sumber	a. Menerima kode sumber berdasarkan jenis bahasa pemrograman dan tipe file b. Memproses kode sumber
3.	Pengelolaan token	a. Memasukan nama file b. Memanipulasi data token
4.	Pencetakan kode sumber	a. Menerima kode sumber yang telah terformat. b. Menampilkan kode sumber yang telah terformat tersebut.

4.3. Deskripsi Data

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

Berikut adalah deskripsi data yang dibutuhkan pada perangkat lunak PuniEdit berupa data tipe file dari bahasa pemrograman, yang akan dijelaskan pada tabel dibawah ini, yaitu:

Tabel 4.2 Deskripsi Data

No.	Jenis Bahasa Pemrograman	Tipe File
1.	68HC11 Assembler	*.hc11;*.asm;*.asc
2.	DSP	*.dsp;*.inc
3.	Baan 4GL	*.cln
4.	Borland Form	*.dfm;*.xfm
5.	C#	*.cs
6.	C/C++	*.c;*.cpp;*.h;*.hpp
7.	Cache	*.mac;*.inc;*.int
8.	CA-Clipper	*.prg;*.ch;*.inc
9.	Cascading Stylesheets	*.css
10.	CPM Reports	*.rdf;*.rif;*.rmf;*.rxf
11.	COBOL	*.cbl;*.cob
12.	CORBA IDL	*.idl
13.	DOT Graph Drawing Description	*.dot
14.	Eiffel	*.e;*.ace
15.	Fortran	*.for
16.	Foxpro	*.prg
17.	Galaxy	*.gtv;*.galrev;*.txt
18.	GEMBASE	*.dml;*.gem
19.	GW-TEL Scripts	*.gws
20.	Haskell	*.hs;*.hws
21.	HP48	*.s;*.sou;*.a;*.hp
22.	HTML	*.htm;*.html
23.	INI	*.ini
24.	Inno Setup Scripts	*.iss
25.	Java	*.java
26.	Javascript	*.js
27.	Kixtart Scripts	*.kix
28.	LEGO LDraw	*.ldr
29.	Modelica	*.mo
30.	Modula-3	*.m3
31.	VB Scripts	*.vbs
32.	MS-DOS Batch	*.bat;*.cmd
33.	Object Pascal	*.pas;*.pp;*.dpr;*.dpk;*.inc
34.	Perl	*.pl;*.pm;*.cgi
35.	PHP	*.php;*.php3;*.phtml;*.inc
36.	Progress	*.w;*.p;*.i
37.	Phyton	*.py
38.	Resource	*.rc
39.	Ruby	*.rb;*.rbw
40.	Semanta	*.sdd

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

No.	Jenis Bahasa Pemrograman	Tipe File
41.	SQL	*.sql
42.	Standard ML	*.sml
43.	Structured Text	*.st
44.	Msg	*.msg
45.	Tcl/Tk	*.tcl
46.	TeX	*.tex
47.	UNIX Shell Scripts	*.sh
48.	Visual Basic	*.bas
49.	Vrml97/X3D World	*.wrl;*.wrml;*.vrl;*.vrml;*.x3d
50.	X86 Assembly	*.asm
51.	XML	*.xml;*.xsd;*.xsl;*.xslt;*.dtd

4.4. Dekomposisi Fisik Modul

Berikut adalah dekomposisi fisik modul pada perangkat lunak PuniEdit, yang akan dijelaskan pada tabel di bawah ini, yaitu:

Tabel 4.3 Dekomposisi Fisik Modul

No	Sub Aplikasi	Modul	Input	Output
1.	Pengelolaan kode sumber	Pengelolaan kode sumber	Nama file, kode sumber	Kode sumber yang telah terformat
2.	Scan kode sumber	Scan kode sumber	Tipe file, jenis bahasa pemrograman, kode sumber	Mengetahui Jenis bahasa pemrograman, jenis token.
3.	Pengelolaan data token	Pengelolaan data token	Nama file, data token	Data token dapat digunakan.
4.	Pembuatan laporan	Pembuatan laporan	Kode sumber yang telah terformat	Sesuai dengan kode sumber yang telah terformat.

4.5. Deskripsi Rinci Modul

Berikut ini adalah deskripsi rinci dari masing-masing modul pada perangkat lunak PuniEdit, yang akan dijelaskan pada tabel di bawah ini, yaitu:

Tabel 4.4. Deskripsi Rinci Modul

No.	Modul	Nama Sub Modul	Deskripsi
1.	Pengelolaan kode sumber	Buka kode sumber	Menerima masukan nama file dari pengguna.
		Edit kode sumber	Melakukan perubahan pada kode sumber, setiap perubahaan akan dilakukan scan kode sumber untuk mendapatkan jenis tokennya.

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

No.	Modul	Nama Sub Modul	Deskripsi
		Buat kode sumber	Menerima jenis bahasa pemrograman, memasukan kode sumber, kode sumber yang dimasukan akan dilakukan scan kode sumber untuk mendapatkan jenis tokennya.
		Simpan kode sumber	Menyimpan kode sumber dalam bentuk file.
2.	Scan kode sumber	Penerimaan tipe file	Mengidentifikasi tipe file masukan untuk mendapatkan jenis bahasa pemrograman yang digunakan.
		Penerimaan jenis bahasa pemrograman	Mengidentifikasi jenis bahasa pemrograman untuk mendapatkan jenis token.
		Penerimaan kode sumber	Menerima kode sumber dan melakukan scan kode sumber untuk mendapatkan jenis tokennya.
		Tampilan jenis bahasa pemrograman	Menampilkan jenis bahasa pemrograman yang teridentifikasi.
		Tampilan jenis token	Menampilkan jenis token yang teridentifikasi
3.	Pengelolaan token	Input data token	Menerima nama file teripe *.hgl, menerima data token.
		Ubah data token	Menerima nama file bertipe *.hgl, memberikan konfirmasi atas perubahan tersebut untuk dilakukan penyimpanan.
		Hapus data token	Memilih nama file bertipe *.hgl, memberikan konfirmasi atas nama file yang akan dihapus tersebut.
4.	Pencetakan kode sumber	Tampilan pencetakan kode sumber yang telah terformat.	Menampilkan kode sumber yang telah terformat.

4.6. Deskripsi Layar

Berikut adalah deskripsi dari layar yang ada pada perangkat lunak PuniEdit, deskripsi-deskripsi tersebut terdiri dari nama layar, deskripsi layar, lay out layar, rancangan layar, deskripsi objek dan jenis dari objek pada rancangan layar perangkat lunak PuniEdit.

4.6.1. Rancangan Layar

Berikut adalah deskripsi masing-masing rancangan layar pada perangkat lunak PuniEdit, yang akan dijelaskan pada tabel di bawah ini, yaitu:

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

Tabel 4.5 Rancangan Layar

No	Nama Layar	Deskripsi Layar	Lay Out Layar	Rancangan Layar
1.	PuniEdit	Halaman utama ketika perangkat lunak dijalankan dan tampilan menu yang dapat digunakan untuk mengelola kode sumber.		
2.	Open File	Halaman untuk membuka kode sumber.		
3.	Save As	Halaman untuk menyimpan kode sumber.		

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

No	Nama Layar	Deskripsi Layar	Lay Out Layar	Rancangan Layar
4.	Syntax source editor	Halaman untuk mengelola bahasa pemrograman
5.	Page setup	Halaman untuk mengatur pencetakan
6.	Print Setup	Halaman untuk mengatur jenis printer yang digunakan
7.	Print Preview	Halaman untuk melihat kode sumber yang telah terformat

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

No	Nama Layar	Deskripsi Layar	Lay Out Layar	Rancangan Layar
8.	Print	Halaman untuk mencetak kode sumber ke media kertas melalui printer
9.	Find	Halaman untuk melakukan pencarian kode sumber
10	Replace	Halaman untuk mencari dan menimpa kode sumber
11	Font	Halaman untuk mengatur jenis huruf

No	Nama Layar	Deskripsi Layar	Lay Out Layar	Rancangan Layar
12	Background	Halaman untuk mengatur warna latar belakang.
13	Character Grid	Halaman untuk menampilkan sekumpulan karakter
14	Converter	Halaman untuk menampilkan konverter.

4.6.2. Deskripsi Proses

Berikut adalah deskripsi dari proses-proses yang ada pada perangkat lunak PuniEdit, deskripsi-deskripsi tersebut terdiri dari nama proses, deskripsi masukan, deskripsi keluaran, dan algoritma dari proses tersebut.

4.2.6.1. Pengelolaan Kode Sumber

Proses pengelolaan kode sumber sebagai pengguna perangkat lunak, memenuhi spesifikasi sebagai berikut:

- Memasukan kode sumber.
- Mengubah kode sumber.
- Menyimpan kode sumber.

4.2.6.2. Scan Kode Sumber

Proses scan kode sumber sebagai pengguna perangkat lunak, memenuhi spesifikasi menerima kode sumber berdasarkan jenis bahasa pemrograman dan jenis tokennya.

4.2.6.3. Pengelolaan Token

Proses pengelolaan token sebagai pengguna perangkat lunak, memenuhi spesifikasi sebagai berikut:

- a. Memasukan nama file.
- b. Mengubah data token.
- c. Menghapus data token.

4.2.6.4. Pencetakan Kode Sumber

Proses Pencetakan kode sumber sebagai pengguna perangkat lunak, memenuhi spesifikasi menerima dan mencetak kode sumber yang telah terformat.

4.6.3. Deskripsi Masukan

Berikut adalah deskripsi masukan dari masing-masing proses pada perangkat lunak PuniEdit, yang akan dijelaskan pada tabel di bawah ini, yaitu:

Tabel 4.7 Deskripsi Masukan

No.	Proses	Masukan
1.	Pengelolaan Kode Sumber	a. Kode sumber b. Nama file c. Tipe file d. Jenis token e. Jenis Bahasa pemrograman
2.	Scan Kode Sumber	a. Kode sumber b. Tipe file c. Jenis bahasa pemrograman
3.	Pengelolaan token	a. Nama file b. Data token
4.	Pencetakan Kode Sumber	Kode sumber yang telah terformat

4.6.4. Deskripsi Keluaran

Berikut adalah dei keluaran masing-masing proses pada perangkat lunak PuniEdit, yang akan dijelaskan pada tabel di bawah ini, yaitu:

Tabel 4.8 Deskripsi Keluaran

No.	Proses	Keluaran
1.	Pengelolaan Kode Sumber	a. Kode sumber b. Tipe file c. Jenis bahasa pemrograman
2.	San Kode Sumber	a. Kode sumber b. Jenis token c. Jenis bahasa pemrograman d. Tipe file
3.	Pengelolaan token	a. Nama file b. Data token
4.	Pencetakan Kode Sumber	Kode sumber yang telah terformat

4.6.5. Algoritma

Berikut ini adalah algoritma untuk proses buka kode sumber dan dilakukan scan kode sumber:

```
Buka_file  
Periksa_Tipe_File  
if bahasa_pemrograman_ketemu then  
    Tampilkan_nama_bahasa_pemrograman  
    Scan_kode_sumber  
else  
    Nama_bahasa_pemrograman_tidak_ditampilkan
```

Berikut ini adalah algoritma untuk proses buat kode sumber dan dilakukan scan kode sumber:

```
if bahasa_pemrograman = pascal then  
begin  
    Highlight ← Bahasa_pemrograman_Pascal  
    if bahasa_pemrograman_ketemu then  
        Tampilkan_nama_bahasa_pemrograman  
    else  
        Nama_bahasa_pemrograman_tidak_ditampilkan  
end
```

5. Implementasi Perangkat Lunak

5.1. Bahasa Pemrograman yang dipilih

Implementasi perancangan perangkat lunak PuniEdit dibuat dengan menggunakan bahasa pemrograman Delphi Versi 7.0. bahasa ini dipilih karena sangat mendukung untuk pembuatan perangkat lunak yang akan dibuat.

Pada dasarnya pemrograman pada Delphi menggunakan bahasa pascal, namun Delphi sudah mendukung terhadap pemrograman berorientasi objek (*Objek Oriented Programming*) yang berada dalam lingkungan system operasi windows, dan bersifat visual.

Untuk perancangan perangkat lunak yang akan dibuat akan lebih mudah karena perancangan antar muka bisa lebih cepat dengan menggunakan fasilitas yang ada atau menggunakan objek-objek yang tersedia.

5.2. Perangkat Lunak dan Perangkat Keras Pendukung

Perangkat lunak yang akan digunakan akan berjalan dengan baik apabila didukung dengan perangkat perangkat keras yang memadai. Agar kedua komponen tersebut bisa saling mendukung maka perlu dirancang dan perlu diperhatikan hal-hal yang sekiranya bisa menjamin kelancaran berjalan sistem yang akan diterapkan.

5.2.1. Perangkat Lunak Pendukung

Perangkat lunak yang akan digunakan adalah:

- a. Sistem Operasi (Windows XP Profesional)
- b. Bahasa Pemrograman (Delphi Versi 7.0)

SynEdit merupakan IDE (*integrated Development Environment*) atau komponen untuk membangun teks editor pada *Windows*, saat ini SynEdit dapat menggantikan peran dari Notepad, hal ini ditunjukkan dengan banyaknya penawaran fitur untuk *programmer*. SynEdit dapat digunakan untuk Borland Dephi, Kylix, dan C++Builder. SynEdit menyediakan *keywords* dan *delimiter* secara lengkap untuk bahasa pemrograman Pascal, C/C++, HTML, ASP, Perl, Java dan banyak lagi bahasa pemrograman lainnya.

SynUniHiglighter merupakan komponen untuk SynEdit yang menyediakan pembuatan/pengelolaan token.

Gambar 5.1. Diagram Kelas SynEdit dan SynUniHighlighter

TCustomControl adalah kelas yang membungkus fungsionalitas untuk melakukan penggambaran terhadap objek Canvas. Umumnya komponen-komponen visual adalah turunan dari TWinControl yang merupakan kelas yang membungkus fungsionalitas untuk *screen object window*, sedangkan komponen-komponen visual yang membutuhkan akses ke objek Canvas tidak diturunkan dari TWinControl melainkan dari TCustomControl. TCustomSynEdit adalah turunan dari TCustomControl karena membutuhkan akses ke objek Canvas, TCustomSynEdit meng-*override* metode paint() dari TCustomControl. TCustomSynEdit mendefinisikan fungsi-fungsi umum penggambaran dan pemrosesan window untuk TSynEdit. TSynEdit adalah kelas turunan dari TCustomSynEdit, TSynEdit memiliki fungsi utama untuk melakukan pemrosesan terhadap teks yang diberikan padanya dan melakukan proses *scanning* token-token yang ada pada teks tersebut. Proses *scanning* token didelegasikan oleh TSynEdit kepada kelas *highlighter*. *Highlighter* ini harus turunan dari kelas TSynCustomHighlighter. Contoh *highlighter* adalah TSynPasSyn untuk highlighter bahasa pemrograman Pascal atau TSynCPPSyn untuk highlighter bahasa pemrograman C++. *Highlighter* juga dapat berupa kelas TSynUniSyn yang merupakan kelas yang mengenkapsulasi fungsionalitas untuk mendefinisikan token-token bahasa pemrograman secara *custom*. TCustomHighlighter mendefinisikan fungsi-fungsi virtual yang harus

diimplementasikan kelas turunannya seperti `GetRange()`, `GetToken()`, `GetTokenPos()` dan sebagainya, sedangkan turunan dari `TCustomHighlighter` harus meng-*override* fungsi-fungsi virtual tersebut dan mendefinisikan token-token spesifik untuk bahasa pemrograman yang dimaksud.

Adapun bahasa pemrograman yang disediakan oleh SynEdit, yaitu: Object Pascal, C/C++, 68HC11 Assembler, DSP, AWK Script, Baan 4GL, Borland Form, C#, Cache, CA-Clipper, Cascading Stylesheets, CPM Reports, COBOL, CORBA IDL, DOT Graph Drawing Description, Eiffel, Fortran, Foxpro, Galaxy, GEMBASE, GW-TEL Script, Haskell, HP48, HTML, INI Files, Inno Setup Scripts, Java, Java Script, KiXtart Scripts, LEGO LDraw, Modelica, Modula-3, VB Script, MS DOS Batch, Perl, PHP, Progress, Phyton, Resource, Ruby, Semanta Data Dictionary, SQL, Standard ML, Structured Text, SynGen Msg, Tcl/Tk, TeX, UNIX Shell Scripts, Visual Basic, Vmrl 97/X3D Word, X86 Assembly, dan XML.

Berikut ini adalah gambar diagram sekuen dari komponen SynEdit:

Gambar 5.2. Diagram Sekuen

Saat user memasukkan input text/kode sumber objek window akan mengirimkan windows message `WM_PAINT` ke komponen SynEdit. SynEdit kemudian akan memanggil metode `paint()` yang di-*override* dari `CustomControl`. Di dalam metode `paint()` ini terdapat fungsi-fungsi untuk menentukan apakah teks yang diinput mengandung token, posisi token tersebut, jenis token, atribut token (warna, style) dan sebagainya. Fungsi ini akan didelegasikan ke *highlighter* yang dipasang pada SynEdit. *Highlighter* ini kemudian akan melakukan proses *scanning* token dan akan

mengembalikan hasilnya ke SynEdit, jika token ditemukan SynEdit akan melakukan proses paint() ke window sehingga akan tampil hasilnya berupa token-token yang di-highlight.

5.2.2. Perangkat Keras Pendukung

Untuk mendukung kebutuhan dari perangkat lunak di atas, maka membutuhkan dukungan perangkat keras minimal sebagai berikut:

1. Processor

Pribadi menggunakan Intel Pentium III – 750 MHz, dan disarankan untuk menggunakan yang lebih besar pertimbangan kebutuhan perangkat keras ini adalah agar perangkat lunak yang dibuat beroperasi dengan baik.

2. Memori primer 128 Mb

Agar saat eksekusi program aplikasi semua unit program yang digunakan pada perangkat lunak tersebut akan di load ke memori utama. Sehingga untuk mengoperasikan perangkat lunak yang mempunyai cukup banyak modul program memerlukan kapasitas memori utama yang cukup besar pula. Maka atas dasar hal tersebut, penulis mencantumkan kebutuhan memori primer sebesar itu.

3. Tempat penyimpanan sekunder minimal berkapasitas 10 Gb

Alasan kebutuhan tempat penyimpanan sekunder dengan kapasitas sebesar itu adalah untuk keperluan penyimpanan data dan kebutuhan perangkat lunak pendukung lainnya.

Untuk mendukung kebutuhan dari perangkat lunak PuniEdit, maka batasan perangkat keras yang digunakan adalah:

- a. Minimal Processor (Pentium)
- b. Memory Primer minimal 16 Mb
- c. Kapasitas ruang kosong tempat penyimpanan sekunder minimal 100 Mb

Kebutuhan dari processor minimalnya adalah Pentium, hal ini karena perangkat lunak yang digunakan akan berjalan dengan optimal apabila kemampuan processor memadai. Memory primer minimal 16 Mb, karena perangkat lunak PuniEdit dapat berjalan pada sistem operasi di atas Windows 95 yang memerlukan penyimpanan memory yang cukup besar untuk melakukan proses-prosesnya.

Kapasitas ruang kosong tempat penyimpanan sekunder minimal 100 Mb, alasan kebutuhan tempat penyimpanan sekunder dengan kapasitas sebesar itu adalah untuk keperluan penyimpanan data.

5.3. Menu Perangkat Lunak PuniEdit

Salah satu interaksi antara user dengan perangkat lunak adalah dengan menggunakan menu, dan biasanya menu ini berupa pilihan-pilihan. Pemakai akan berinteraksi dengan memilih menu yang ada untuk masuk ke dalam sistem. Oleh karena itu, agar perangkat lunak PuniEdit bisa lebih mudah dipahami maka disusun menu yang bisa mewakili semua kegiatan yang ada didalam sistem. Menu yang dirancang yaitu:

Tabel 5.1 Daftar Menu Perangkat Lunak PuniEdit

No.	Menu	Sub Menu	Fungsi
1.	File	New	Membuat kode sumber baru.
		Open	Membuka file.
		Save	Menyimpan kode sumber.
		Save As	Menyimpan kode sumber dengan nama file yang berbeda.
		Page Setup	Mengatur layout kode sumber ketika akan dilakukan pencetakan melalui media kertas.
		Print Setup	Mengatur jenis print yang akan digunakan.
		Print Preview	Melihat hasil cetakan sebelum mencetaknya.
		Print	Mencetak kode sumber melalui media kertas.
		Export	Mentransfer tipe file yang sedang aktif ke dalam format HTML/RTF.
		Close All	Menutup semua kode sumber yang sedang dibuka.
		Close	Menutup kode sumber yang sedang aktif.
		Recent File	Menampilkan nama file yang telah dibuka sebelumnya
		Clear Recent File	Membersihkan nama file yang telah dibuka sebelumnya
2.	Edit	Exit	Keluar dari program.
		Undo	Kembali ke perintah sebelumnya
		Redo	Kembali ke perintah selanjutnya.
		Cut	Memindahkan kode sumber yang diblok.
		Copy	Menggandakan kode sumber yang diblok.
		Paste	Menampilkan kode sumber yang ada di clipboard.
		Delete	Menghapus kode sumber yang sedang diblok.
		Select All	Memblok seluruh kode sumber.
3.	Search	Time/Date	Menyisipkan waktu dan tanggal.
		Find	Melakukan pencarian karakter terhadap kode sumber.
		Find Next	Melakukan pencarian karakter selanjutnya
		Find Previous	Melakukan pencarian karakter sebelumnya
		Replace	Melakukan pencarian karakter dan mengganti hasil pencarian karakter tersebut.
		Count Char	Menghitung karakter

PEMBANGUNAN PERANGKAT LUNAK PUNIEDIT
(Perangkat Lunak Editor untuk Multi Language Programming)
Budiman

No.	Menu	Sub Menu	Fungsi
4.	Format	Font	Mengatur jenis huruf
		Background	Memberi warna latar pada area kerja
5.	Tools	Scan Token	Mengelompokkan jenis token.
		Character Set	Menampilkan kumpulan karakter.
		Mathematical Expression Interpreter	Menampilkan program matematika.
		View Date/Time	Menampilkan waktu dan tanggal.
		DOS	Memanggil program MS-DOS
		Internet Explorer	Memanggil program Internet Explorer
		Syntax Source Editor	Pengelolaan token
		Load Syntax Source	Menjalankan data token yang telah dibuat.
6.	Window	Cascade	Menampilkan seluruh anak program tersusun secara bertumpuk.
		Tile Vertical	Menampilkan seluruh anak program tersusun secara vertikal.
		Tile Horizontal	Menampilkan seluruh anak program tersusun secara horisontal.
		Arrange Icon	Merapikan tata letak anak program dalam kondisi minimize.
		Minimize All	Menampilkan seluruh anak program dalam kondisi minimize.
		Maximize All	Menampilkan seluruh anak program dalam ukuran maksimal.
		Restore All	Menampilkan seluruh anak program dari kondisi minimize.
		Close All	Menutup seluruh anak program
7.	Help	PuniEdit Help	Menampilkan bantuan perangkat lunak PuniEdit.
		Borland Delphi	Menampilkan bantuan Borland Delphi
		Turbo Pascal for Windows	Menampilkan bantuan Turbo Pascal
		Turbo C++ for Windows	Menampilkan bantuan Turbo C++
		About PuniEdit	Menampilkan tentang perangkat lunak PuniEdit
		About Your Computer	Menampilkan tentang komputer yang sedang digunakan.

KESIMPULAN DAN SARAN

6.1. Kesimpulan

Dari hasil perancangan dan implementasi dapat diperoleh beberapa kesimpulan yaitu:

1. Perangkat lunak PuniEdit yang dibangun dalam tugas akhir ini merupakan perangkat lunak *text editor* untuk kode sumber.
2. Perangkat lunak PuniEdit dapat melakukan pemeriksaan bahasa pemrograman yang digunakan.
3. Perangkat lunak PuniEdit dapat melakukan pemeriksaan jenis token dari kode sumber dengan memberikan warna teks yang berlainan.
4. Perangkat lunak PuniEdit dapat melakukan pengelolaan token.
5. Perangkat lunak PuniEdit dapat melakukan pencetakan kode sumber melalui media kertas.

6.2. Saran

Saran-saran untuk pengembangan tahap selanjutnya adalah sebagai berikut:

1. Perangkat lunak PuniEdit yang dibangun dalam tugas akhir ini hanya dapat melakukan *scan token*. Oleh karena itu untuk pengembangan selanjutnya harus bisa dibuat *compiler*-nya.
2. Perangkat lunak PuniEdit yang dibangun dalam tugas akhir ini tidak dapat melakukan *setting* atau pengaturan warna dari setiap token. Oleh karena itu untuk pengembangan selanjutnya harus bisa dibuat *setting* warna token.

DAFTAR PUSTAKA

- [UTD01] Utdiratatmo, Firrar : “*Teknik Kompilasi*”, J&J Learning Yogyakarta, 2001.
- [PRE02] Pressman, Roger S : “*Software Engineering A Practitioner’s Approach*”, McGraw-Hill, 2002.
- [SAN92] Santoso, Insap : “*Struktur Data Menggunakan Turbo Pascal 6.0*”, Andi Yogyakarta, 1992.
- [MUN99] Munir, Rinaldi : “*Algoritma & Pemrograman*”, Informatika Bandung, 1999.
- [KOI00] Koiso, Hideo : “*Provides an HTML highlighter for SynEdit*”,
<http://SynEdit.SourceForge.net>, 2000.
- [TRI01] Trier, Michael : “*Provides a C++ syntax highlighter for SynEdit*”,
<http://SynEdit.SourceForge.net>, 2001.
- [WAL01] Waldenburg, Martin : “*Provides a Pascal/Delphi syntax highlighter for SynEdit*”,
<http://SynEdit.SourceForge.net>, 2001