A STUDY OF FORMAL PARALLEL LANGUAGE GENERATION

By

GREGORY R. GUDENBURR

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1980

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
August 1998

OKLAHOMA STATE UNIVERSITY

A STUDY OF FORMAL PARALLEL LANGUAGE GENERATION

Thesis Approved:

Mansur Samadzadeh

Thesis Advisor

D. E. Hedman

Blayne E. Mayfield

Wayne B. Powell

Dean of the Graduate College

PREFACE

This purpose of this study was to develop a tool to automate the generation of sentences of a grammar. The generation of the sentences were to be done in parallel. The parallel method used is based on Dijkstra's guarded commands. The tool can be used as a teaching aid to check the existence of a sentence in a language generated by a grammar.

The software tool that was developed as part of this thesis work places certain restrictions on the type of grammar being used. These restrictions limit the languages to be considered to those that are recursive and do not contain empty productions. These restrictions only allow the productions of the grammar to increase the sentential forms in size or to leave the size of the sentential forms unchanged. All derivations were done in a leftmost manner. The tool was written to run under Microsoft Windows NT as an MFC-based application.

ACKNOWLEDGMENTS

I sincerely need to thank my advisor Dr. Mansur Samadzadeh for his guidance, help, thoroughness, and patience during this project. The other members of my thesis committee, Drs. Blayne Mayfield, and George Hedrick, also need to be mentioned for their support and understanding during this project.

My mother also provided encouragement in completing my graduate studies. I need to thank two friends, Lee Fields and Christy Valliere for their time, friendship, and companionship during the long drives to and from Stillwater, in addition to a study group that we had together.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

To date there are limited tools available that can be used to perform derivation testing for grammars. Finding a derivation path for a sentence presents limited options and hand processing the grammar can be difficult with a large complicated grammar. With large grammars, the number of potential paths can further complicate this process. More complication arises when the membership of a sentence in a language is in question.

The tool provided as part of this thesis is a prototype tool, which can be used to automate the derivation path discovery. This tool will, within a defined set of constraints, perform an exhaustive search of the derivation paths for a sentence. This search will determine the membership of the sentence in the language generated by a grammar and the corresponding derivation path.

The sequential method of producing a derivation path has been extended through the use of parallel guarded commands. By modifying parallel guarded commands, the processing of a grammar can now be done in parallel to produce multiple derivation paths concurrently.

The rest of this thesis is organized as follows: Chapter II discusses some of the common parallel language constructs in use today. Chapter III introduces the sequential and

parallel versions of guarded commands. Chapter IV defines grammars and introduces the derivation process. Chapter V evaluates the tool developed as part of this thesis. Finally, Chapter VI summarizes the work and suggestions for future work.

CHAPTER II

DISCUSSION OF PARALLEL LANGUAGE CONSTRUCTS

2.1 Definitions and Background

Concurrency in a program can be performed either on a single processor using multiprogramming or on multiple processors. With the use of multiple processors, it is possible to execute statements in a program simultaneously therefore reducing the overall execution time as compared to executing them on a single processor [Paterson and Silberscha 85].

Two of the popular methods used for specifying concurrency in programming are the language constructs `fork/join` and `cobegin/coend`. These two language constructs have been used in several languages in various forms for providing concurrency.

Programs that are run with concurrent language constructs must produce the same results as functionally equivalent programs that do not contain these constructs. Many potential issues can be introduced when concurrent programming constructs are used. Some of these issues include race conditions for data structures and resource deadlocks. This thesis work does not address these issues other than to note them.

2.2 fork/join

The fork/join programming construct was introduced as an extension to the "goto" language command [Hoare 85]. There have been two major versions of the fork/join construct since its introduction. The major difference in the two versions is where the two paths of execution continue after the fork. In the following discussion, the term thread will be used to indicate that the fork language construct can cause either the creation of two paths of execution within a single process space or the creation of two independent process spaces.

The first version, fork I, transfers control to label I, which becomes a new execution thread and the original execution thread continues execution with the language construct following the fork. This allows for the execution of the program in two locations: one starting at label I and the other continuing execution with the statement following the fork. Additionally, both threads may execute the fork command again [Hoare 85].

The opposite of the fork is the join language construct. This construct is used to merge two or more execution threads into a single execution thread. The first thread reaching the join language construct must wait until the other thread reaches the same join. When both threads have reached the join, a single execution thread continues execution.

Line	Language Statements
1	<some language construct >
2	fork L;
3	a = a + 1;
4	goto J:
5	L: b = b + 10;
6	J: join;

Figure 1. Original `fork/join` Example

An example using the original `fork/join` is given in Figure 1 above showing a single execution thread beginning at line 1. With the execution of line 2, two concurrent execution blocks are started: one beginning at line 3 and the other at line 5. Both code blocks are executed concurrently (either actually or in a multiplexed manner) until each reaches the `join` construct in line 6. The first execution thread to reach line 6 waits for the second thread to reach the `join`. When both threads have reached the `join` statement, a single execution thread continues with the statement following line 6.

A variant of the original `fork/join` exists today on Unix. The Unix `fork` creates two processes in a parent/child relationship. The child is a copy of the parent at the time the `fork` construct is executed. Both the parent and child continue execution with the statement following the `fork`. The return code from the `fork` is used to determine if execution is occurring in the parent or child. Instead of the `join` construct, the Unix version of `fork` uses a `wait` construct. This is used only by the parent to determine when the child has ended execution.

Line	Language Statements
1	<some language construct >
2	int status;
3	pid = fork();
4	if (pid == 0)
5	{
6	a = a + 1;
7	exit(0);
8	}
9	b = b + 10;
10	wait(&status);

Figure 2. UNIX `fork` Example

An example using the Unix `fork` is shown in Figure 2 above showing a single execution thread beginning at line 1. With the execution of line 2, the parent processes is copied to a new process and both processes start executing at line 3. The "process copy" includes both the code and any local/global variables and their values at the time the `fork` is executed. Line 3 allows the process to identify itself as parent or child. The returned value from the `fork()` is the process identification (PID) of the child. If the PID is equal to zero, the process is the child; otherwise the process is the parent.

If the process is the child, lines 6 and 7 are executed with line 7 terminating the execution of the child process. Note that the value of variable "a" updated in the child process is not reflected in the parent process.

If the process is the parent, execution continues with line 9. The `join` construct used in the previous figure has been replaced with a `wait` construct. The `wait` construct is used to determine when the child process has ended. The parent process will wait at line 10 until the

child process has ended execution and then the parent process continues processing with the language statements following line 10. After the `wait` statement is executed, only one execution path continues.

2.3 `cobegin/coend`

The `cobegin` and `coend` (or `parbegin` and `parend`) language constructs were proposed by Dijkstra [Hoare 85]. The `cobegin` construct ensures that different blocks of code in the body of the program are executed concurrently. All code blocks are processed independently until they reach the `coend` construct. When all blocks have reached the `coend`, a single execution path continues.

By ensuring that each code block is different much of the housekeeping and concern about which execution thread is executing is eliminated. It is much less likely that two or more threads may accidentally execute the same code, as can happen with the `fork`. Additionally, the `cobegin/coend` constructs can be nested to allow a parallel block to execute the `cobegin/coend` constructs.

Line	Language Statements
1	<some language construct >
2	cobegin;
3	{ a = a + 1; }
4	{ b = b + 10; }
5	coend;

Figure 3. cobegin/coend Example

An example using cobegin/coend is shown in Figure 3 above. Again a single execution thread exists at line 1. When line 2 is executed, both code blocks at lines 3 and 4 begin execution in parallel. Both code blocks execute independently with each stopping execution when they reach line 5. When both code blocks have reached line 5, execution continues as one thread at the statement following line 5.

2.4 Other Constructs

Other constructs have been devised that address the need for concurrency differently as in the case of OCCAM [Dowsing 88]. OCCAM uses the constructs PAR and SEQ to control concurrency. To begin concurrent execution, the PAR statement is used. The PAR construct is similar to cobegin discussed before. To define sequential execution blocks within a parallel statement, the SEQ statement is used. Language statements within a SEQ block are executed sequentially.

CHAPTER III

GUARDED COMMANDS

3.1 Introduction

Dijkstra introduced the concept of "guarded commands" in a paper published in 1975 [Dijkstra 75]. This work provided an alternative and a repetitive construct, to control execution conditions when they could not be determined from the initial state. Hoare used this initial work to extend it to operate as a parallel construct instead of just a sequential language construct [Hoare 78]. Hoare's work is the basis of this project.

3.2 Guarded Commands

Dijkstra worked with "guarded commands" to solve programming problems where the execution conditions could not be determined from the initial state [Dijkstra 75]. This work resulted in several language constructs. These language constructs contained boolean conditions called guards that had to evaluate to true before the corresponding statement lists would be executed.

The statement list contained in each guard is executed left to right when the controlling guard is evaluated to true. The execution of each guarded command is sequential. The syntax of a guarded command follows [Dijkstra 75]:

- <guard command set> ::= <guarded command> { [] <guarded command> }
- <guarded command> ::= <guard> → <guard list>
- <guard> ::= <boolean expression>
- <guard list> ::= <statement> {; <statement> }
- <alternative construct> ::= **if** <guarded command set> **fi**
- <repetitive construct> ::= **do** <guarded command set> **od**
- <statement> ::= <alternative construct> | <repetitive construct> | "other statements"

3.3 Parallel Guarded Commands

Hoare combined Dijkstra's work with guarded commands and the `cobegin/coend` constructs to change the execution of the statement list. This new work changed the execution of the statement list from a sequential left to right execution to parallel. The commands in the statement list are executed simultaneously and the statement list ends only when all commands have ended [Hoare 78]. Hoare changed the syntax of the guarded commands slightly to account for the addition of the parallel constructs. The new syntax follows:

- <parallel command> ::= [<process>{||<process>]
- <process> ::= <process label><command list>
- <process label> ::= <empty>|<identifier>::<identifier>(<label subscript>{,statement> {< label subscript >})
- < label subscript > ::= < integer constant >|<range>
- < integer constant > ::= <numeral>|< bound variable >
- < bound variable > ::= <identifier>
- <range> ::= <bound variable>:<lower bound>,,<upper bound>
- <lower bound> ::= <integer constant>

- <upper bound> ::= <integer constant>
- <command list>::={<declaration>;|<command>;}<command>
- <command>::<simple command>|<structured command>
- <simple command>::<nullcommand><assignment command>|<input command>|<output command>
- <structured command>::=<alternative command>|<repetitive command>|<parallel command>
- <null command>::=skip
- <command list>::={<declaration>;|<command>;}<command>

3.4 Examples

In this section a simple example of both the sequential and the parallel guarded commands is included. First the example of the sequential guarded commands is given followed by the parallel version of the same command.

An example of a guarded command can be found in Figure 4. This illustrates an alternative construct statement. This alternative construct consists of a guarded command set containing two guarded commands. Each guarded command contains a boolean guard, which must be true before the guard list is executed. In this example, the first guard is evaluated `(x > y)` and, if true, the statement list is executed. If the first statement list is executed, variable `m` is assigned the value of variable `x`. Execution continues with the evaluation of the second guard `(y >= x)` and, if true, the statement list is executed. If the second statement list is executed, variable `m` is assigned the value of variable `y`.

```
if x > y → m:=x [] y >= x → m:=y fi.
```

Figure 4. Example - Guarded Command

An example of a parallel guarded command can be found in Figure 5. This illustrates the parallel alternative construct corresponding to Figure 4, which was sequential. The parallel version of the alternative construct consists of the same guarded command set containing two guarded commands now called a command list. Each command still contains a boolean guard, which must be true, before the command list is executed. In this example however, both boolean guards are evaluated simultaneously. If the first guard (x > y) is evaluated to be true, the statement list is executed assigning variable m the value of variable x. If the second guard (y >= x) is true, the statement list is executed assigning variable m the value of variable y. Execution will continue at the statement following the parallel guarded command when both statements have completed execution.

```
[x > y → m := x [] y >= x→ m := y]
```

Figure 5. Example - Parallel Guarded Command

CHAPTER IV

LANGUAGE GENERATION

4.1 Grammars

A formal language can be generated by a grammar by recursively and selectively applying the rules of the grammar to the start symbol until only terminal characters exist in the resulting string. A string that contains only terminal characters is called a sentence. An intermediate result containing both terminal and non-terminal characters is referred to as a sentential form of the grammar.

Grammars are classified according to the format or type of their productions. A grammar is defined [Aho and Ullman 72] by a 4-tuple $G=(N, \Sigma, P, S)$ where:

1. N is a finite set of non-terminal symbols.
2. Σ is a finite set of terminal symbols, disjoint from N.
3. P is a finite subset of $(N \cup \Sigma)^* N (N \cup \Sigma)^* X (N \cup \Sigma)^*$. An element of (α,β) in P will be written $\alpha \rightarrow \beta$ and is called a production.
4. S is a distinguished symbol in N called the start symbol.

Based on the format of the productions, grammars are classified into four types. The four types are Right Linear, Context Free, Context Sensitive, and Unrestricted. The format of the productions classifies grammars as follows.

1. Right Linear: If each production in P is of the form $A \rightarrow xB$ or $A \rightarrow x$, where A and B are in N and x is in Σ^*.
2. Context Free: If each production in P is of the form $A \rightarrow \alpha$, where A is in N and α is in $(N \cup \Sigma)^*$.
3. Context Sensitive: If each production in P is of the form $\alpha \rightarrow \beta$, where $|\alpha| <= |\beta|$.
4. Unrestricted: If there are no restrictions on the productions.

4.2 Derivation Process

The process of applying the rules of a grammar repeatedly produces a sequence of strings. These resulting strings are not necessarily distinct. The sequence of strings starting at α and ending with β is represented as $\alpha \rightarrow^* \beta$, meaning that this is a sequence of strings such that $\alpha = \alpha_0, \alpha_{i-1} \rightarrow \alpha_i$ for $1 <= I <= k$ and $\alpha_k = \beta$. This is called a derivation process [Aho and Ullman 72].

Determining if a sentence exists in the language generated by a given grammar can be done in two ways or directions. The first method applies the rules of the grammar from the start symbol in an exhaustive manner until the sentence can be proven to exist or not in the language generated by the grammar. The second method starts with the sentence and applies the rules of the grammar in a parsing manner attempting to reduce the sentence to the start symbol. The second method is similar to the work done by a parser.

In the derivation process, applying the rules of a grammar can be a nontrivial task. Given a long sentence, it can be complex and time consuming (depending on the grammar) to determine the derivation path of the sentence. In the process of determining the derivation path of a given sentence, many paths may be produced that may not result in the given sentence.

Two questions must be answered about the derivation process. First, is the problem decidable and second, if decidable, what is the complexity of the derivation process.

Given a language L(G) that is generated from a grammar G, is answering the question of membership for a word or sentence X in L(G) decidable? The answer to this question is yes as long as the language is recursive. Recursive languages include regular languages, context free languages (CFL), and context sensitive languages (CSL) without empty productions.

The question of complexity of the derivation process is ignored at the present because this study is an initial feasibility study and not an attempt to invent/define an efficient derivation process.

To simplify the derivation process, two additional assumptions are made by this project. First, all substitutions are done with the leftmost match being substituted. Second, since the grammars used in this project are ever increasing in length, the derivation process can be terminated when a sentential form exceeds the length of the ending sentence.

An example of a derivation follows. A simple grammar is given as: G=({E,T,F},{a, +, *, (.)}, P, E) with productions: $E \rightarrow E+T \mid T$, $T \rightarrow T*F \mid F$, $F \rightarrow (E) \mid a$ [Aho and Ullman 72]. To find the sentence a+a*a in the language generated by the grammar G starting from the start symbol E, a derivation path of: $E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T*F \Rightarrow a+F*F \Rightarrow a+a*F \Rightarrow a+a*a$ can be generated. There are several decision points in the derivation process that are necessary to generate the correct derivation path for the sentence.

CHAPTER V

EVALUATION

5.1 Sample Programs

The grammars chosen for testing the derivation tool were selected from textbooks. If a selected grammar contained the empty production, it was modified to include the empty production as a terminal character. The test grammars attempt to show that the tool correctly performs the derivation process. The test cases selected include both sentences and sentential forms from the chosen grammars.

First a step by step run will show the execution of the tool with a test grammar. This step by step run will show the execution of the tool (called PLUG) developed as part of this thesis. The test grammar is a modified grammar taken from Aho and Ullman's book [Aho and Ullman 72]. The grammar is given below.

$S \rightarrow 0A1$
$S \rightarrow 00A1$
$S \rightarrow e$

Starting PLUG will produce the MDI mainframe as shown in Figure 6 below. The mainframe includes three menu options: File, View, and Help. The File menu selection is

used to open, save, and print a guarded grammar (see Appendix B for explanation). The View menu is used to control the display of the Status bar and Tool bar, and the Help menu displays the program version. The Tool bar can be used for quick selection of the file and edit functions.

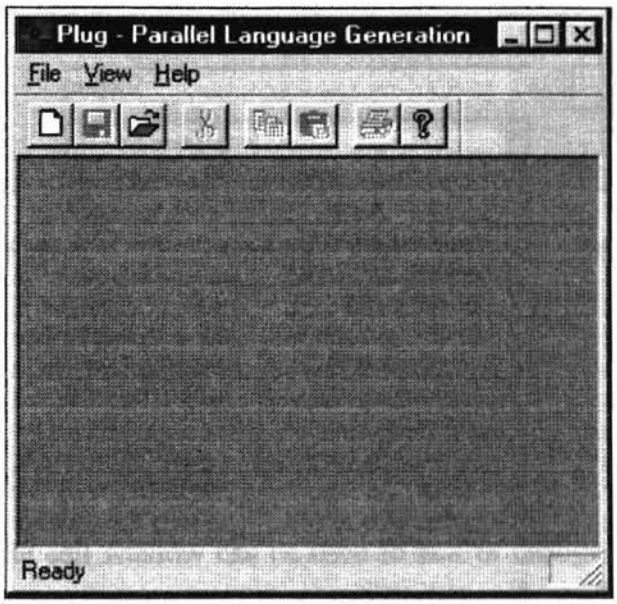

Figure 6. PLUG MDI main window

Selecting the file→open menu option will cause the file open dialog box to be displayed as in Figure 7. The dialog box shown is one of the common dialog boxes used by Windows. For this example, select the file `page87.grm` and click the "Open" button. The test grammar will be loaded into an edit window.

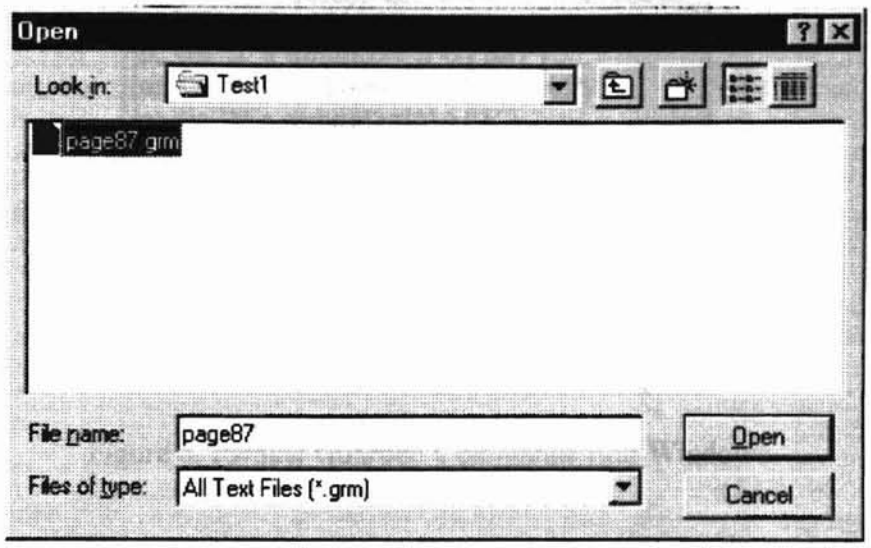

Figure 7. PLUG Common File Open Dialog Box

The test guarded grammar `page87.grm` is loaded into an edit window as shown in Figure 8 below. The edit window can be used to edit or change the guarded grammar. Once the guarded grammar is loaded, the `edit` and `process` menu bar options are available. The edit menu gives access to the edit functions `undo`, `copy`, `cut`, and `paste`. The `process` option is used to syntax check the guarded grammar in the selected edit window and to specify processing options. The format of the guarded grammar shown in Figure 8, is discussed in detail in Appendix B.

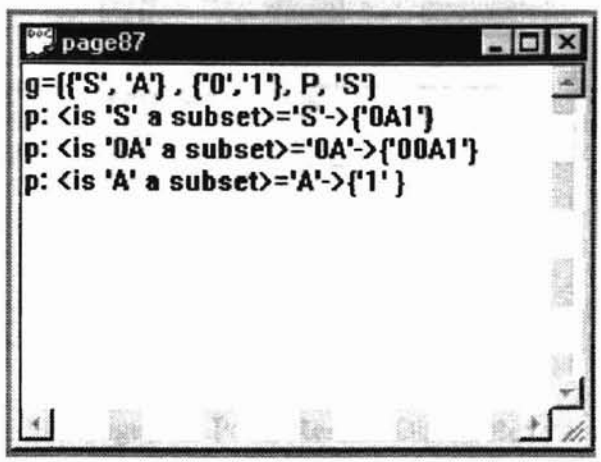

Figure 8. Parallel Guarded Commands Edit Windows

The guarded grammar must be syntax checked before the derivation process can begin. To syntax check the guarded grammar, select menu option: process ➔ syntax check (this menu item is made available when the grammar is loaded). The syntax checking performed ensures that all terminal and non-terminal characters used in the production rules are included in the grammar definition. Checking is also performed to make sure that the guard for each production rule matches the left hand side of the production rule. The results of syntax checking the guarded grammar statements from Figure 8 are shown in Figure 9. Figure 9 shows the number of lines syntax checked and the number of syntax errors.

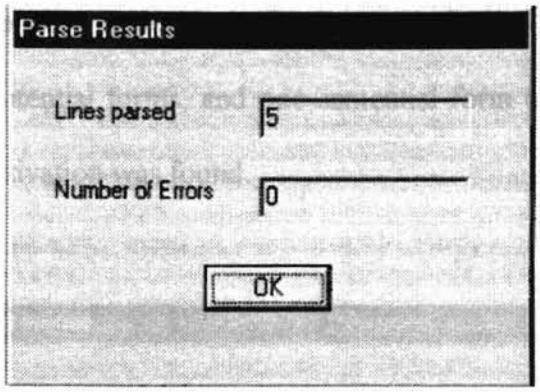

Figure 9. Parse Results Dialog Box

After the 'OK' button is clicked in Figure 9, a new window is displayed with the results of the syntax check. This window will show any syntax errors that were discovered. In this case, there are no syntax errors and the results window will look the same as the edit window in Figure 8.

If syntax errors were detected, the location of the syntax error would have been indicated by line number and column. Along with this indicator, additional text diagnostic messages would be given to assist in correcting the error(s).

Now that the guarded grammar has been syntax checked, processing can begin. To begin processing, select the process ➔ process menu selection. The dialog box depicted in Figure 10 will be shown. Fill in the "Starting From" and the "Search For" strings. In this case, the strings 'S' and '0001111' were selected, respectively. After filling, in the required fields click the 'Start' button in Figure 10.

After the derivation process ends, the processing results and statistics are shown in Figure 10. A total of eight threads were started, the maximum number of concurrent threads was seven, and the processing time was less than one second.

The derivation process created an eight-node derivation tree. This tree included three sentences, five sentential forms, and one sentential form terminated due to length, and the "Search For" derivation was found.

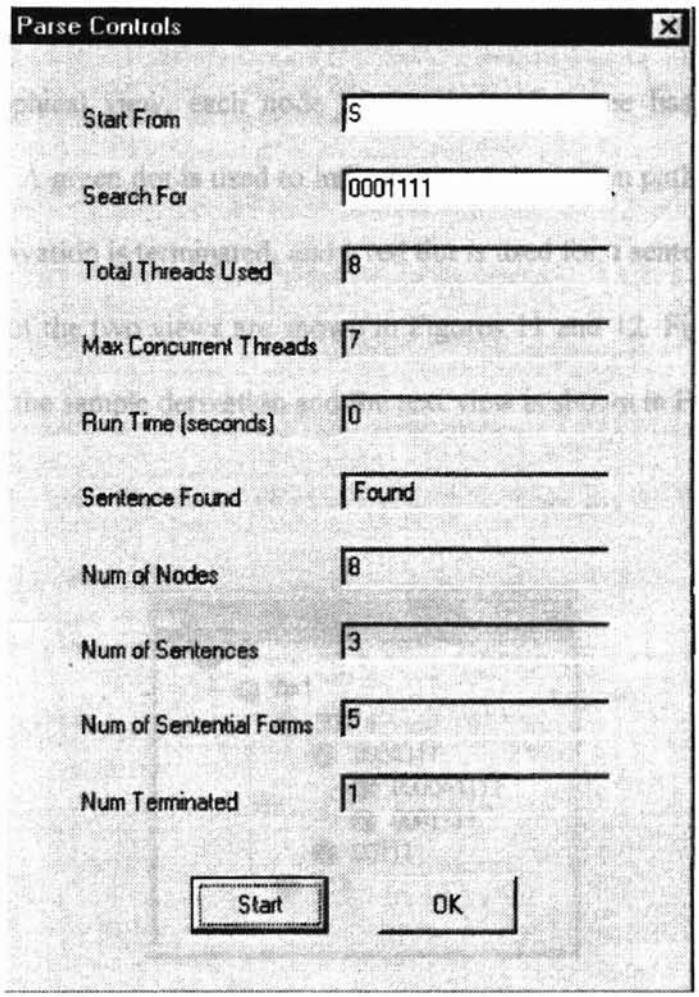

Figure 10. Processing Results Dialog Box

When the 'OK' button in Figure 10 is clicked, two results windows are openned. The first results window is a text listing of the derivation paths, and the second is a graphical view of the derivation paths. In both windows, the terminated derivations are

distingushed from sentence derivation, to assist in easy identification.

In the window showing the text listing of the derivations, the terminated sentential forms and sentences are distinguished by appending an indicator. Appending a '(T)' character string to the derivation indicates that it was terminated. Sentences are indicated by appending an '(S)' character string to the derivation.

In the graphical view, each node of the derivation tree has a color indicator showing its status. A green dot is used to indicate that a derivation path continues, a black dot is used if a derivation is terminated, and a red dot is used for a sentence.

Examples of the two views are shown in Figures 11 and 12. Figure 11, shows the graphical view of the sample derivation and the text view is shown in Figure 12.

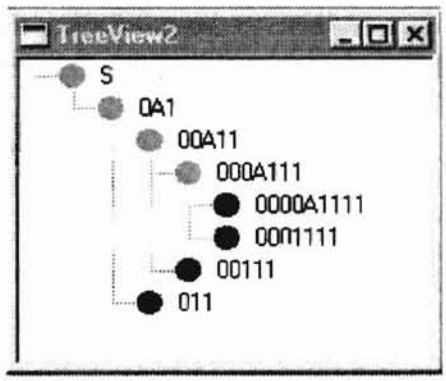

Figure 11. Tree View of Processing

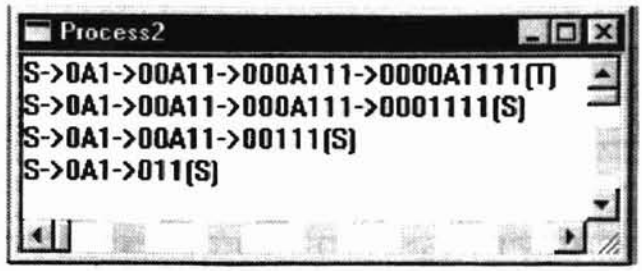

Figure 12. Text View of Processing

Figure 13 shows the test guarded grammars used for validating the tool (see Appendix B for explaination of the format of these entries). Each guarded grammar is assigned a number that is used for identification in Figures 14 and 15. Eleven grammars were selected for validating the tool. The grammars used in numbers 4.1, 4.2, 7.1, and 7.2 were modified to include the empty production as a terminal production. To ensure that the tool can locate multiple derivation paths for a sentence, grammars were also selected that contained ambiguity. The test grammars were selected from books written by Aho and Ullman [Aho and Ullman 72] [Aho and Ullman 78].

Test Number	Guarded Grammar
1	g=({'S', 'A'} , {'0','1'}, P, 'S') p: <is 'S' a subset>='S'->{'0A1'} p: <is '0A' a subset>='0A'->{'00A1'} p: <is 'A' a subset>='A'->{'1' }
2.1 2.2 2.3	g=({'E', 'T','F'} , {'a','+','*','(',')'}, P, 'E') p: <is 'E' a subset>='E'->{'E+T'|'T'} p: <is 'T' a subset>='T'->{'T*F'|'F'} p: <is 'F' a subset>='F'->{'(E)' | 'a' }
3.1 3.2 3.3	g=({'S', 'A','B', 'C'} , {'a','b','c'}, P, 'S') p: <is 'S' a subset>='S'->{'aSBC'|'abC'} p: <is 'CB' a subset>='CB'->{'BC'} p: <is 'bB' a subset>='bB'->{'bb'} p: <is 'bC' a subset>='bC'->{'bc'} p: <is 'cC' a subset>='cC'->{'cc'}
4.1 4.2	g=({'S'} , {'0','1','e'}, P, 'S') p: <is 'S' a subset>='S'->{'0S'|'1S' | 'e'}
5.1 5.2	g=({'S', 'A', 'B'} , {'0','1'}, P, 'S') p: <is 'S' a subset>='S'->{'A0B'|'B1A'} p: <is 'A' a subset>='A'->{'BB'|'0'} p: <is 'B' a subset>='B'->{'AA'|'1'}
6.1 6.2	g=({'S', 'B', 'C'} , {'a','b', 'c'}, P, 'S') p: <is 'S' a subset>='S'->{'abC'|'aB'} p: <is 'B' a subset>='B'->{'bc'} p: <is 'bC' a subset>='bC'->{'bc'}
7.1 7.2	g=({'S', 'A', 'B'} , {'0','1', 'e'}, P, 'S') p: <is 'S' a subset>='S'->{'0A'|'1S' | 'e'} p: <is 'A' a subset>='A'->{'0B' | '1A'} p: <is 'B' a subset>='B'->{'0S' | '1B'}
8.1 8.2	g=({'A'} , {'a','b'}, P, 'S') p: <is 'A' a subset>='A'->{'aaAAA' | 'abA'|'b'}
9.1 9.2	g=({'S', 'A', 'B'} , {'a','b'}, P, 'S') p: <is 'S' a subset>='S'->{'aAB' | 'BA'} p: <is 'A' a subset>='A'->{'BBB' | 'a'} p: <is 'B' a subset>='B'->{'AS' | 'b'}
10.1 10.2 10.3	g=({'S', 'A', 'B'} , {'a','b'}, P, 'S') p: <is 'S' a subset>='S'->{'AB'} p: <is 'A' a subset>='A'->{'Aa' | 'bB'} p: <is 'B' a subset>='B'->{'a' | 'Sb'}
11.1	g=({'S', 'C'} , {'i','t', 'e', 'a', 'b'}, P, 'S') p: <is 'S' a subset>='S'->{'iCtS'|'iCtSeS'|'a'} p: <is 'C' a subset>='C'->{'b' }

Figure 13. Test Grammars

Figure 14 shows the statistics for the derivation process for each test case. Test cases were selected that resulted in the derivation path being discovered as well as those that did not. The statistic table shows the following:

- Test: A number referring to the guarded grammar, shown in Figure 13.
- Start From: Starting sentential form.
- Search For: Target sentential form of the derivation process.

- Total Thrds: Total number of threads started.
- Max Thrds: Maximum number of threads run concurrently.
- Run Time: Run time in seconds.
- Found: Indicator if a derivation for "Search For" was found.
- Num Nodes: Total number of nodes in the derivation tree.
- Num Sentences: Total number of sentences in the derivation tree.
- Num Sentential: Total number of sentential forms in the derivation tree.
- Num Term: Number of derivations terminated.

Normally, the number of nodes in the derivation tree matched the number of threads started; however, in case 4.1 this is not true. The number of threads started is larger in this case because 1,849 threads had to be restarted. The reasons why the threads had to be restarted are discussed in the next section.

Test	Start From	Search For	Thrds	Max thrds	Run Time	Found	Num Nodes	Num Sentences	Num Sentential	Num Term
1	S	110011	6	3	0	No	6	2	1	N/A
2.1	E	a+a*a	1035	265	4	Yes	1035	133	902	485
2.2	E	a+a	65	21	0	Yes	130	18	112	58
2.3	E	a++a	65	19	1	No	65	9	56	29
3.1	S	Aabbcc	11	5	0	Yes	11	2	9	1
3.2	S	abc	4	3	0	Yes	4	1	3	1
3.3	S	cba	4	3	0	No	4	1	3	1
4.1	S	0101010101e	7992	2144	84	Yes	6142	2047	4095	2048
4.2	S	1e1	22	18	0	No	22	15	7	8
5.1	S		363	124	1	Yes	363	68	295	152
5.2	00AA	000AAB	103	36	0	Yes	103	19	84	19
6.1	S	abc	5	4	0	Yes	5	2	3	0
6.2	S	cbaa	5	3	0	No	5	2	3	0
7.1	S	0101e	73	44	0	No	73	10	63	32
7.2	S	000e	36	22	0	Yes	36	5	31	16
8.1	A	aabbb	19	9	0	Yes	19	4	15	9
8.2	A	Aabbabb	49	19	1	Yes	49	9	40	24
9.1	S	bbbb	139	40	1	Yes	139	20	119	67
9.2	aAB	baba	25	14	0	No	25	2	23	15
10.1	S	Baabaab	2311	917	29	Yes	2311	110	2201	1319
10.2	S	bBABb	176	80	0	Yes	176	9	167	92
10.3	S	Baabaab	63	33	1	Yes	63	5	58	30
11.1	S	Ibtibtaea	190	76	1	Yes	190	36	154	84

Figure 14. Test Results

Figure 15 shows the derivation paths discovered by the program while testing. If a test lists more than one derivation path, the grammar is ambiguous. The test numbers correspond to the guarded grammar numbers used in Figure 13. Only the grammars that did discover the "Search For" strings are listed.

Test	Derivation Paths
1	A⇒00A1⇒00A11⇒000111
2.1	E⇒E+T⇒T+T⇒T*F+T⇒F*F+T⇒a*F+T⇒a*F+F⇒a*a+F⇒a*a+a E⇒E+T⇒T+T⇒T*F+T⇒F*F+T⇒a*F+T⇒a*a+T⇒a*a+F⇒a*a+a E⇒E+T⇒T+T⇒T*F+T⇒F*F+T⇒F*F+F⇒a*F+F⇒a*a+F⇒a*a+a E⇒E+T⇒T+T⇒T*F+T⇒T*a+T⇒F*a+T⇒F*a+F⇒a*a+F⇒a*a+a E⇒E+T⇒T+T⇒T*F+T⇒T*a+T⇒F*a+T⇒a*a+T⇒a*a+F⇒a*a+a E⇒E+T⇒T+T⇒F+T⇒F+T*F⇒F+F*F⇒a+F*F⇒a+a*F⇒a+a*a E⇒E+T⇒T+T⇒F+T⇒F+T*F⇒a+T*F⇒a+F*F⇒a+a*F⇒a+a*a E⇒E+T⇒T+T⇒F+T⇒F+T*F⇒a+T*F⇒a+T*a⇒a+F*a⇒a+a*a E⇒E+T⇒T+T⇒F+T⇒a+T⇒a+T*F⇒a+F*F⇒a+a*F⇒a+a*a E⇒E+T⇒T+T⇒F+T⇒a+T⇒a+T*F⇒a+T*a⇒a+F*a⇒a+a*a E⇒E+T⇒E+T*F⇒T+T*F⇒F+T*F⇒F+F*F⇒a+F*F⇒a+a*F⇒a+a*a E⇒E+T⇒E+T*F⇒T+T*F⇒F+T*F⇒a+T*F⇒a+F*F⇒a+a*F⇒a+a*a E⇒E+T⇒E+T*F⇒T+T*F⇒F+T*F⇒a+T*F⇒a+T*a⇒a+F*a⇒a+a*a E⇒E+T⇒E+T*F⇒T+T*F⇒T+T*a⇒F+T*a⇒F+F*a⇒a+F*a⇒a+a*a E⇒E+T⇒E+T*F⇒T+T*F⇒T+T*a⇒F+T*a⇒a+T*a⇒a+F*a⇒a+a*a E⇒E+T⇒E+T*F⇒E+F*F⇒T+F*F⇒F+F*F⇒a+F*F⇒a+a*F⇒a+a*a E⇒E+T⇒E+T*F⇒E+F*F⇒T+F*F⇒T+a*F⇒F+a*F⇒a+a*F⇒a+a*a E⇒E+T⇒E+T*F⇒E+F*F⇒T+F*F⇒T+a*F⇒T+a*a⇒F+a*a⇒a+a*a E⇒E+T⇒E+T*F⇒E+F*F⇒E+a*F⇒T+a*F⇒F+a*F⇒a+a*F⇒a+a*a E⇒E+T⇒E+T*F⇒E+F*F⇒E+a*F⇒T+a*F⇒T+a*a⇒F+a*a⇒a+a*a E⇒E+T⇒E+T*F⇒E+F*F⇒E+a*F⇒E+a*a⇒T+a*a⇒F+a*a⇒a+a*a E⇒E+T⇒E+T*F⇒E+T*a⇒T+T*a⇒F+T*a⇒F+F*a⇒a+F*a⇒a+a*a E⇒E+T⇒E+T*F⇒E+T*a⇒T+T*a⇒F+T*a⇒a+T*a⇒a+F*a⇒a+a*a E⇒E+T⇒E+T*F⇒E+T*a⇒E+F*a⇒T+F*a⇒F+F*a⇒a+F*a⇒a+a*a E⇒E+T⇒E+T*F⇒E+T*a⇒E+F*a⇒T+a*a⇒F+a*a⇒a+a*a E⇒E+T⇒E+T*F⇒E+T*a⇒E+F*a⇒E+a*a⇒T+a*a⇒F+a*a⇒a+a*a
2.2	E⇒E+T⇒T+T⇒F+T⇒F+F⇒a+F⇒a+a E⇒E+T⇒T+T⇒F+T⇒a+T⇒a+F⇒a+a E⇒E+T⇒E+F⇒T+F⇒F+F⇒a+F⇒a+a E⇒E+T⇒E+F⇒T+F⇒T+a⇒F+a⇒a+a E⇒E+T⇒E+F⇒T+F⇒T+a⇒F+a⇒a+a
3.1	S⇒aSBC⇒aabBC⇒aabBCC⇒aabbCC⇒aabbcC⇒aabbcc
3.2	S⇒aSBC⇒abc
4.1	S⇒0S⇒01S⇒010S⇒0101S⇒01010S⇒010101S⇒0101010S⇒01010101S⇒010101010S⇒010101 0101S⇒0101010101e
5.1	S⇒A0B⇒A01⇒BB01⇒1B01⇒1AA01⇒10A01⇒1000
5.2	S⇒00AA⇒000A⇒000BB⇒000AAB
6.1	S⇒abC⇒abc S⇒aB⇒abc
7.2	S⇒0A⇒00B⇒000S⇒000e
8.1	A⇒aaAAA⇒aabAA⇒aabbA⇒aabb
8.2	A⇒aaAAA⇒aabAA⇒aabbA⇒aabbabA⇒aabbabb
9.1	S⇒BA⇒Ba⇒bBBB⇒bbBB⇒bbbB⇒bbbb
10.1	A⇒AB⇒AaB⇒AaSb⇒AaABb⇒bBaABb⇒baaABb⇒baabBBb⇒baabaBb⇒baabaab
10.1	S⇒AB⇒ASb⇒bBSb⇒bBABb
10.3	S⇒AB⇒bBB⇒baB⇒baSb
11.1	S⇒iCtS⇒ibtS⇒ibtiCtSeS⇒ibtibtSeS⇒ibtibtaeS⇒ibtibtaea

Figure 15. The Derivation Paths

5.2 Evaluation and Observations

The derivation process for a sentence in a language generated by a grammar can produce a large number of derivations in effort to determine the correct derivation path of the sentence. The methods used in this project to limit the number of active derivations did reduce the number of derivation paths. Terminating the derivation of a sentential form, when its length exceeded the target sentence, was used to limit the number of active derivations.

Two simple tests were used to determine if processing could be ended for a derivation path. The first test terminated the derivation process when the sentential form was a sentence and the second test terminated the derivation process when the length of the sentential form exceeded the length of the target sentence. The tool terminated processing when all derivations either were sentences or had terminated.

The guarded grammar statements used to define a grammar were designed to be simple and reflect the four-tuple definitions for a grammar. These statements take their basis from Dijkstra's guarded commands.

One area where the tool could be made more efficient is handling thread startup and termination. The overhead of starting and terminating as many threads as was necessary for this project resulted in a considerable amount of overhead. Rearchitecting this area to reduce overhead will need to take into account the fact that threads use shared memory.

One limitation that was discovered during testing was that Windows NT seemed to have a limit of about 1,600 threads per process. When this limit was reached, the AfxBeginThread API, which is used to start each thread, returned a NULL pointer. When the

NULL pointer was returned, a new thread was not started. No documentation was found to indicate what the limit was for the number of threads per process. The tool worked around this limitation by adding a restart queue. The restart queue was used to restart any thread that failed to start when this limitation was reached.

Finally, the tool produced as part of this thesis work was able to successfully process and work on all of the test grammars.

CHAPTER VI

SUMMARY AND FUTURE WORK

6.1 Summary

The objective of this thesis was to develop a tool to test the feasibility of automating the generation of the derivation paths of sentences in a grammar. This generation process was done using a parallel construction method. All relevant paths were tried in parallel in an effort to discover the derivation path of a given sentence.

This derivation method can be used in either a forward or backward direction. The forward method starts from a given sentential form (usually the start symbol) of the grammar. The rules of the grammar are applied until a match is found or all derivation paths end without matching the ending sentence. The backward direction works like the forward direction, but starts with the ending sentence and works backward to the start symbol.

This thesis is based on work done by Hoare and Dijkstra with guarded commands. Hoare's proposed extensions to guarded commands was the basis for this thesis and was used in controlling the derivation process.

6.1 Future work

Future work may include making the tool more language aware so that other types of languages and grammars can be supported. Adding more instrumentation to the code could show in more detail what is happening during the derivation process. Changing the language to JAVA would allow usage over the Internet and possible more machine independence. Using LEX or YACC for parsing could allow for more flexibility in parsing the guarded grammar as well as reducing the parsing complexity. Finally, work on the efficiency of the thread usage would improve the performance of the tool.

REFERENCES

[Aho and Ullman 72] Alfred V. Aho and Jeffery D. Ullman, *The Theory of Parsing Translation,and Compiling, Volume 1: Parsing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.

[Aho and Ullman 78] Alfred V. Aho and Jeffery D. Ullman, *Principles of Compiler Design*, Addison-Wesley Publishing Company, Reading, MA, 1978.

[Blelloch 96] Guy E. Blelloch, "Programming Parallel Algorithms", *Communications of the ACM*, Vol. 39, No. 3, pp. 85-97, March 1996.

[Dijkstra 75] Edsger W. Dijkstra, "Guarded Commands, Nondeterminacy, and Formal Derivation or Programs", *Communications of the ACM*, Vol. 18, No. 8, pp. 453-457, August 1975.

[Dowsing 88] Roy Dowsing, *Introduction to Concurrency Using Occam*, Nostrand Rienhold, London, UK, 1988.

[Hoare 78] C. A. R. Hoare, "Communicating Sequential Processes", *Communications of the ACM*, Vol. 21, No. 8, pp. 666-677, August 1978.

[Hoare 85] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985

[Kahn 74] Gilles Kahn, "The Semantics of a Simple Language for Parallel Programming", *Information Processing 74, Proceedings of the IFIP Congress 1974*, North-Holland Publishing Company, pp. 471-475, 1974.

[Paterson and Silberschatz 85] James L. Peterson and Abraham Silberschatz, *Operating System Concepts*, Addison-Wesley Publishing Company, Reading, MA, 1985.

APPENDIX A: GLOSSARY

Grammar:	A formal system for defining a language.
Language:	A set of strings generated from an alphabet by applying a set of productions or re-writing rules.
MFC:	Microsoft Foundation Classes
Sentence:	A result of the application of one or more productions of a grammar to the start symbol containing only terminal characters.
Start Symbol:	A special non-terminal character in a grammar.

APPENDIX B: TOOL INFORMATION AND CODE LISTING

In this appendix, the code for the project is shown. A software tool called PLUG (Parallel Language Generation) was written as a Windows NT MFC application. This application uses several classes to produce the derivation path of a sentence. The data flow from parsing the guarded grammar keywords to producing the derivation is described below.

The first step is to parse the guarded grammar keywords. The grammar class is used to parse the keywords and produce the structures necessary to control the derivation process. The derivation process is controlled by the guarded commands described in the grammar. During parsing, the character sets that make up the non-terminal, terminal, and designated non-terminal start symbol are collected. Additionally, the production rules of the grammar are also collected.

The data structures used to save the non-terminal, terminal, start symbol, and productions are the CharString class and the list array template. The productions are also stored in a structure, which contains the guard and each production rule.

After the guarded grammar keywords have been parsed, processing is ready to begin. The first step in processing is to request the input of the starting and ending sentential forms. The starting sentential form is used as the starting point for the derivation

process and the ending sequential form is the target of the derivation process.

A SententialForm class is used to store the derivation starting point. The SententialForm class also inherits the CharString class and extends it by adding flags to describe the state of the sentential form. The flag indicates the type of sentential form as either a sentence or a terminated sentential form.

Next, the root node of the derivation tree is allocated. The root node contains the starting point for the derivation. Each node of the derivation is an instance of the derivation class. Each node in the derivation tree points to all possible subsequent derivations. The derivation class includes methods to change and set the next node pointers.

Once the root derivation is initialized, the derivation process starts with the initiation of a thread. The thread begins by scanning the guards contained in the grammar for a guard contained in the current sentential form. For each matching guard, a new thread is started. This new thread allocates a new derivation as a child of the previous derivation and performs the character substitutions per matching rule in the grammar. The process continues recursively until all derivation paths end in sentences or are terminated.

A derivation path is terminated whenever the length of a sentential form exceeds the length of the "Search For" string. Each derivation either ends in a sentence or the "Search For" string is found.

Two guarded grammar statements are used to describe the grammar and the parallel guarded commands associated with the grammar. The two statements are the grammar definition and production definition cards. Each are discussed in more detail in

the following paragraphs.

This grammar control statement defines the grammar to be processed. The information required is the characters making up both the terminal and non-terminal character sets. Additionally, the designated non-terminal character is also defined as the start symbol. The third operand is a placeholder for the grammar productions, which will be defined in the production statements. The syntax of the statement is defined as:

g=({'N...N',...'N...N'}, {'Σ ...Σ',...,'Σ...Σ'}, P, 'S')

{'N...N',...'N...N'} is a comma delimited list of characters enclosed in single quotes. These characters make up the non-terminal characters of the grammar. Each character must be enclosed within a single quote.

{'Σ ...Σ',...,'Σ...Σ'} is a comma delimited list of characters enclosed in single quotes. These characters make up the terminal characters of the grammar. Each character must be enclosed within a single quote.

P is a placeholder for the production runs.

S is the designated character from the non-terminal character set called the start symbol. This character must be enclosed in single quotes.

The production statement defines the guarded production rules of the grammar. If the guard is true, each possible production is processed in parallel. The guard is fired for the leftmost match of the sentential form currently being processed. The syntax of the statement is defined as:

P: <is 'X' a subset >= 'X' -> {'y...y' | 'z...z' | ...}, where

P: identification that the control statement is a production.

<is 'X' a subset > Is a singly quoted string consisting of characters from the terminal and non-terminal sets. The guard is true for the first occurrence of the 'X' within the current sentential form being processed. If the guard is not true, the production rule is ignored and processing continues with the next production rule.

'X' is a singly quoted string consisting of characters from the terminal and non-terminal sets. This character string is replaced to form the new production from the production rules of this control statement.

Some basic code characters are: 28 header files, 26 cpp files, lines 8,075, number of comments 2,300, and the number of MFC files 24. The number of classes used is 19.

In the code listing that follow, the code for the various files and module appear in the following order:

```
CharString.cpp
ChildFrm.cpp
Derivation.cpp
Grammar.cpp
MainFrm.cpp
ParseControls.cpp
Plug.cpp
Plug.rc
PlugEdit.cpp
PlugEditDoc.pp
PlugParallel.cpp
PlugParseDialog.cpp
PlugParseOut.cpp
PlugParseOutDoc.cpp
PlugProcessOut.cpp
PlugProcessOutDoc.cpp
PlugStatistics.cpp
PlugTreeView.cpp
PlugTreeViewDoc.cpp
SententialForm.cpp
StdAfx.cpp
CharString.h
ChildFrm.h
Derivation.h
Grammar.h
List.h
MainFrm.h
ParseControls.h
Plug.h
PlugEdit.h
PlugEditDoc.h
PlugParallel.h
PlugParseDialog.h
PlugParseOut.h
PlugParseOutDoc.h
PlugProcessOut.h
PlugProcessOutDoc.h
PlugStatistics.h
PlugTreeView.h
PlugTreeViewDoc.h
Resource.h
SententialForm.h
StdAfx.h
Cont.ico
Plug.ico
Plug.rc2
PlugDoc.ico
Stop.ico
Term.ico
Tollbar.bmp
ReadMe.txt
Plug.mak
```

CharString.cpp / CharString.h

The implementation/header file for this class provides a class that is similar to the ANSI C template string class. This class was written instead of using the ANSI template class because all of the platforms that were considered initially for this project did not have the standard class library available. This class implements several string operators used to manipulate a character string. These operations include assignment, concatenation, and index overloaded operations. Additional operations are also provided that return the character string, length, and provide an index function. During development of this class, overloaded input and output operators were used to test this class but are not used in the thesis itself.

```
#ifndef MY_CHARSTRING
#define MY_CHARSTRING

//
// CharString class is used to manage a description CharString.  Also CharString provides
// overload operators =, +, <, >, ==, [], >>, and <<. The overloaded operators
// are used to manage the CharStrings.
//

class CharString {


  protected:

        int len;              // length of character string.
    char *str;      // pointer to the character string.

  public:

        //
        // Constructors for the character string.
        //
        CharString(int n=0);
    CharString(const char *s);
    CharString(const CharString &s);
    ~CharString();

        //
        // Overloaded operators.
        //    The overloaded operators are used to manipulate the character
        //    string.
        //
    CharString& operator= (const CharString &s);
    CharString& operator= (const CharString *s);
    CharString operator+ (const CharString &s);
    CharString operator+ (const int &i);
    int operator>  (const CharString &s);
    int operator<  (const CharString &s);
    int operator== (const CharString &s);
    char& operator[] (int n);

        //
        // misc operators.
        //
        char* GetChar();                        // return the character string.
        int length(void);                       // return the length of the character
string
        int Contains(const CharString &s);      // Does this character string contain
                                                //    the
parameter character string as
                                                //    as a
substring.
```

```
        //
        // overloaded input/output operators
        //
        friend ostream& operator<< (ostream& out, CharString &s);
    friend istream& operator>> (istream& in,  CharString &s);

};

#endif



//
// CharString Class methods
//
//  These methods are used as a replacement for the MFC String class or the ANSI
//  string class.  At the time that this project was started the execution platform
//  was not firm and a decision to develop my own string class was made. A standard
//  string class should be used.

#include "stdafx.h"

#if !defined(true)
#define true 1
#endif

#if !defined(false)
#define false 0
#endif



#include <string.h>
#include <iostream.h>
#include <iomanip.h>
#include "CharString.h"

#ifdef _AFX
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#endif

//
// CharString default constructor. Allocates a string of "n" characters. The
// character string is then set the zero lenght.
//
CharString::CharString(int n)
{
  str = new char [n+1];
  str[0] = '\0';
  len = n;
}

//
// CharString default constructor. Allocates a character string and copies the
// passed string into the newly allocated string.
//
CharString::CharString(const char *s)
{
  len = strlen(s);
  str = new char [len + 1];
  strcpy(str,s);
}

//
// CharString default copy constructor. Makes sure and make a new copy of
// the string pointed to by the string pointer.
//
CharString::CharString(const CharString &s)
{
  len = s.len;
  str = new char [len + 1];
  strcpy(str,s.str);
```

```
}

//
// CharString default deconstructor. delete the storage obtained for Str.
//
CharString::~CharString()
{
  delete [] str;
}

//
// Overload '=' operator.  The '=' operator copies one string class to
// another.  The old string destination is delete and reallocated to
// the size of the new string.
//
CharString&
CharString::operator= (const CharString &s)
{
  len = s.len;
  char *p = new char[len+1];
  strcpy(p,s.str);
  delete str;
  str = p;
  return *this;
}

//
// Overload '=' operator.  The '=' operator copies one string class to
// another.  The old string destination is delete and reallocated to
// the size of the new string. The source is a CharString pointer.
//
CharString&
CharString::operator= (const CharString *s)
{
  if ( !s ) return (CharString) NULL;
  len = s->len;
  char *p = new char[len+1];
  strcpy(p,s->str);
  delete str;
  str = p;
  return *this;
}

//
// Overload '+' operator.  The '+' concatinates two string characters
// and returns the resulting string.
//
CharString
CharString::operator+ (const CharString &s)
{
  CharString result;

  delete [] result.str;
  result.len = s.len + len;
  result.str = new char [result.len + 1];
  strcpy(result.str,str);
  strcat(result.str,s.str);

  return result;
}

//
// Overload '+' operator.  The '+' concatinates string characters and an integer
// character returns the resulting string.
//
CharString
CharString::operator+ (const int &s)
{
  CharString result;

  delete [] result.str;
  result.len = 1 + len;
  result.str = new char [result.len + 1];
  strcpy(result.str,str);
  result.str[result.len-1] = s;
  result.str[result.len]=0;

  return result;
```

```
}

//
// Overload '<' operator.  The '<' operator compares two string characters
// and returns the result.
//
int
CharString::operator< (const CharString &s)
{
  return(strcmp(str,s.str)<0);
}

//
// Overload '>' operator.  The '>' operator compares two string characters
// and returns the result.
//
int
CharString::operator> (const CharString &s)
{
//  cout << "operator>" << "str = " << str << " s.str = " << s.str << endl;
  return(strcmp(str,s.str)>0);
}


//
// Return the character string contained in the class.
//
char*
CharString::GetChar()
{
  return str;
}

//
// Overload '==' operator.  The '==' operator compares two string characters
// and returns the result.
//
int
CharString::operator== (const CharString &s)
{
  if ( len != s.len ) return false;
  return(strcmp(str,s.str)==0);
}

//
// Overload '[]' operator.  The '[]' returns the character in the string at
// offset [x]
//
char&
CharString::operator[] (int elem)
{
  static char temp = '\0';
  if (elem < 0 || elem > len) return temp;
  return str[elem];
}


//
// Does this CharString contain the parameter CharString.
//
int
CharString::Contains(const CharString &s)
{
        char *c = NULL;

        c = strstr(str,s.str);
        if ( c == NULL ) return -1;

        return (c - str);
}



//
// return the length of this CharString
//
int
CharString::length(void)
```

```
{
        return len;
}

//
// Overload '<<' operator.  The '<<' operator will print the string str.
//
ostream&
operator<< (ostream &out, CharString &s)
{
  out << s.str;
  return out;
}

//
// Overload '>>' operator.  The '>>' read a string and inputs it into
// string.
//
istream&
operator>> (istream &in, CharString &s)
{
  const int Max = 81;
  char temp[Max], *p;

  // read in string description
  cin.getline(temp,Max);

  // find first non blank character in string
  p = temp;
  while(*p == ' ' && *p != '\0') p++;

  // load the string data into the string class
  delete [] s.str;
  s.len = strlen(p);
  s.str = new char[s.len+1];
  strcpy(s.str,p);

  return in;
}
```

ChildFrm.cpp / ChildFrm.h

This set of header/implementation files are used to describe the child frame in the Windows MDI application. These files are as generated by the class wizard.

```
// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHILDFRM_H__E69D48FE_4843_11D1_A845_00C0D1095F74__INCLUDED_)
#define AFX_CHILDFRM_H__E69D48FE_4843_11D1_A845_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CChildFrame : public CMDIChildWnd
{
        DECLARE_DYNCREATE(CChildFrame)
public:
        CChildFrame();

// Attributes
public:

// Operations
public:
```

```
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CChildFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CChildFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
        //{{AFX_MSG(CChildFrame)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_CHILDFRM_H__E69D48FE_4843_11D1_A845_00C0D1095F74__INCLUDED_)


// ChildFrm.cpp : implementation of the CChildFrame class
//
// MFC child frame for MFC application.

#include "stdafx.h"
#include "plug.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
        //{{AFX_MSG_MAP(CChildFrame)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
{
        // TODO: add member initialization code here

}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CMDIChildWnd::PreCreateWindow(cs);
```

```
}

/////////////////////////////////////////////////////////////////////////////
// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
        CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
        CMDIChildWnd::Dump(dc);
/////////////////////////////////////////////////////////////////////////////
// CChildFrame message handlers
}

#endif //_DEBUG
```

Derivation.cpp / Derivation.h

The derivation class is used to describe a derivation of a grammar. Each derivation is a node in the derivation tree. Each node contains pointers to the next node (child) in the derivation process. Methods have been provided to Add, and Get the children of this derivation. A method is also provided to return the state of the derivation.

```
#ifndef DERIVATION
#define DERIVATION

//
// Derivation class is used to define a sentenial for of the grammar.  Each Derivation
points to
// the next derivation of the grammar.
//
// DATA:
//  str       - pointer to character string.
//      NextDerivations - pointer to Derivations that are children of this Derivation
//  Num_Derivations - Number of children this Derivation points to.
//      Max_Derivations - Maximum number of Derivation NextDerivations can contain without
reallocation.

//
#include "List.h"
class SententialForm;

class Derivation {

  protected:

        SententialForm          data;
        ListArray<Derivation>     Children;
        int                                     Num_Children;
        int                                     Max_Children;
        int                                     Derivation_Level;

        void AddNodetoTree(CTreeCtrl *m_ctlTreeCtrl,
                           TV_INSERTSTRUCT &tv,
                                   char *Reverse,
                                   int Cont_Icon_offset,
                                   int Stop_Icon_offset,
                                   int Term_Icon_offset);



  public:

//
```

```
//  Initialization Contructor/Destructor routines
//
    Derivation(SententialForm &s, int Level);
       Derivation();
    Derivation(const Derivation &n);
    ~Derivation();

//
//  Manipulation routines
//
       int Add_Child(Derivation &n);                          // Add Derivation "N" as a
child of this Derivation
       int Add_Child(Derivation &n, int index);     // Add Derivation "N" as a child of
this Derivation

       //  at offset index.
       int SetMax(int index);                          // Add Derivation "N" as a child of
this Derivation
       int Get_Level(void);                                          // Get the level of
this Derivation
       int Get_Child(Derivation *n, int Start);     // Return a pointer to this
derivations children
       char* Get_String(void);                      // Return the sentential form for the
derivation

       void Get_Tree_Stats(int &NumNodes,
                           int &NumSententialForms,
                                 int &NumSentences,
                                 int &NumTerminated);       // Get the Tree
stats

//
//  I/O routines
//
       void Print_Tree(ostream &out, SententialForm CS);     // Print the tree to the
output stream
    void Print_Tree_To_StringArray(CStringArray &out,     // Print the tree to a MFC
                                     SententialForm CS);    //
String Array.
    void Print_Tree_To_Tree(CTreeCtrl *m_ctlTreeCtrl,     // Print the tree to the
                            TV_INSERTSTRUCT tv,                          //    Output
Tree.
                                                HTREEITEM &t,
                                                int Cont_Icon_offset,
                                                int Stop_Icon_offset,
                                                int Term_Icon_offset);

    int  Print_Tree_To_Tree_Reverse(char *Reverse,        // print the tree to the
                                    CTreeCtrl *m_ctlTreeCtrl, // output tree with
                                                TV_INSERTSTRUCT &tv,
    // Reverse being the node.
                                                HTREEITEM &t,
                                                int Cont_Icon_offset,
                                                int Stop_Icon_offset,
                                                int Term_Icon_offset);


       friend ostream& operator<< (ostream& out, Derivation &n); // overloaded output
operator

};

#endif
```

Grammar.cpp / Grammar.h

The grammar header/implementation class files are used to describe the grammar being processed by this program. The grammar class contains the terminal, non-terminal, and productions of the grammar. A method to parse and syntax check statements is provided

by the class. Syntax checking a keyword statement adds the appropriate data structures to the class to allow the derivation process to be done. Additional class methods are provided to search the guards for a match.

```
//////////////////////////////////////////////////////////////////////
//
// Grammar.h: interface for the Grammar class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_GRAMMAR_H__8CB47537_284B_11D1_A813_00C0D1095F74__INCLUDED_)
#define AFX_GRAMMAR_H__8CB47537_284B_11D1_A813_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "List.h"

//
// Structure for describing the guarded commands.
//
        typedef struct GuardStruct {
                CharString                              Guard;
                ListArray<CharString>  Statement;
                int                                             NumStatements;
        } GuardStruct;

class Grammar
{


protected:
        ListArray<CharString>  NT;              //  The nonterminal CharString array.
        ListArray<CharString>  T;               //  The terminal CharString array.
        int                                             NumNT;          //  The number of
non-terminal CharStrings
        int                                             NumT;           //  The number of
terminal CharStings.
        int                                             NumProductions;         //  The number
of productions
        ListArray<GuardStruct> GuardedCommands;//  The production of the grammar.

        #define                                 GRAMMARFSAMAXX 51
        #define                                 GRAMMARFSAMAXY 256
        short unsigned int              GrammarFSA[GRAMMARFSAMAXX][GRAMMARFSAMAXY]; // The
parse FSA
    unsigned char                       ValidCharacters[256]; // The valid characters used
by the grammar

public:

        //
        // The constructor destructors for the grammar class
        Grammar();
    ~Grammar();

        int Parse(const char *input, CharString &results);                      // Parse the
input line
        int     MaxChildren(void);
                // Return the max children
        int     AddNT(CharString &CS);
        // Add a non-terminal
        int isNT(CharString &CS);
        // Is the CharString a Non-Terminal
        int     AddT(CharString &T);
        // Add a terminal
        int isT(CharString &CS);
        // Is the CharString a terminal
        int     AddProduction(CharString &CSGuard);
        // Add a production Guard
        int AddProduction(int GuardNum, CharString &CSGuardCmd);     // Add a production at
Guard num
```

```
        int ScanGuardForMatch(char *Current, int index);            // Scan the guards for
a guard match
        int NumGuards(void) { return NumProductions; }                                    //
Return the number of productions
        int NumGuardProductions(int index);
        // Return the number of productions for this guard
        int ScanProduction(CharString &input,                       // Scan the
productions at rulenum and for production
                                    CharString &output,
        // replacing the characters.
                                    const int RuleNum,
                                    const int ProductionNum);
        int isSentence(CharString input);                           // is the charstring
(input) a sentence.

    friend ostream& operator<< (ostream& out, Grammar &s); // overload the output operator
};

#endif // !defined(AFX_GRAMMAR_H__8CB47537_284B_11D1_A813_00C0D1095F74__INCLUDED_)


// Grammar.cpp: implementation of the Grammar class.
//
//////////////////////////////////////////////////////////////////////


#include "stdafx.h"


#include <string.h>
#include <iostream.h>
#include <iomanip.h>
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include "CharString.h"
#include "Grammar.h"

#ifdef _AFX
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#endif

#ifndef true
#define true 1
#endif
#ifndef false
#define false 0
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Grammar::Grammar()
{
        int i, j;

        for ( i = 0; i < 256; i++ )
                ValidCharacters[i]=i;

        NumNT = NumT = NumProductions = 0;  // initialize the number of non-terminals,
                                                                        // terminals,
and number of productions
                                                                        // to zero.

        //
        // set everthing to invalid
        //
        for (i = 0; i <         GRAMMARFSAMAXX; i++)
                for ( j = 0; j < GRAMMARFSAMAXY; j++ )
                    GrammarFSA[i][j] = 99;   // invalid state

        //
```

```
        // State (0) find the G
        //
        GrammarFSA[0]['g']=1;  // "g" go to state 1
        GrammarFSA[0]['G']=1;  // "g" go to state 1
        GrammarFSA[0]['p']=24;  // "P" go to state 24
        GrammarFSA[0]['P']=24;  // "p" go to state 24
        GrammarFSA[0][' ']=0;  // ignore the blanks

    // Parse g=({'n',,,}, {'x',,,} ,p , 's' )
        GrammarFSA[1]['=']=2;                                   // "=" go to state 2
        GrammarFSA[1][' ']=1;                                   // ignore the blanks

        GrammarFSA[2]['(']=3;                                   // "(" go to state 3
        GrammarFSA[2][' ']=2;                                   // ignore the blanks

        GrammarFSA[3]['{']=14;                                  // "{" go to state 3
        GrammarFSA[3][' ']=3;                                   // ignore the blanks

        // State (4)

        GrammarFSA[5][',']=6;                                   // "," go to state 6
        GrammarFSA[5][' ']=5;                                   // ignore the blanks

        GrammarFSA[6]['{']=18;                                  // "{" go to state 7
        GrammarFSA[6][' ']=6;                                   // ignore the blanks

        // State (7)

        GrammarFSA[8][',']=9;                                   // "," go to state 9
        GrammarFSA[8][' ']=8;                                   // ignore the blanks

        GrammarFSA[9]['p']=10;                                  // "P" go to state 10
        GrammarFSA[9]['P']=10;                                  // "P" go to state 10
        GrammarFSA[9][' ']=9;                                   // ignore the blanks

        GrammarFSA[10][',']=11;                                         // "," go to
state 11
        GrammarFSA[10][' ']=10;                                         // ignore the
blanks

        GrammarFSA[11]['\'']=12;                                // "," go to state 11
        GrammarFSA[11][' ']=11;                                         // ignore the
blanks

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[12][i] = 12;
        GrammarFSA[12]['\\']=22;                                // ignore the blanks
        GrammarFSA[12]['\'']=13;                                // ignore the blanks

        GrammarFSA[13][' ']=13;                                         // ignore the
blanks
        GrammarFSA[13][')']=23;                                         // ignore the
blanks

        GrammarFSA[14]['\'']=15;                                // Find a starting '
        GrammarFSA[14][' ']=14;

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[15][i] = 15;
        GrammarFSA[15]['\'']=17;                                // Find an ending '
        GrammarFSA[15]['\\']=16;                                // find an escape
character

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[16][i] = 15;
        GrammarFSA[17][',']=14;                                         // Find an
ending ,
        GrammarFSA[17]['}']=5;                                  // Find the end of the
NT set.

        GrammarFSA[18]['\'']=19;                                // Find a starting '
        GrammarFSA[18][' ']=18;

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[19][i] = 19;
        GrammarFSA[19]['\'']=21;                                // Find an ending '
        GrammarFSA[19]['\\']=20;                                // find an escape
character

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[20][i] = 19;
```

```
        GrammarFSA[21][',']=18;                                          // Find an
ending ,
        GrammarFSA[21]['}']=8;                                        // Find the end of the
NT set.

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[22][i] = 12;

        // END state
        GrammarFSA[23][' ']=23;          // Find an ending ,

        // Parse p: < 'x' is a subset> = 'x'-> { 'yy..' | 'xx..' ...}
        GrammarFSA[24][':']=25;          // Find ':'
        GrammarFSA[24][' ']=24;          // Find ' '

        GrammarFSA[25]['<']=26;          // Find '<'
        GrammarFSA[25][' ']=25;          // Find ' '

        GrammarFSA[26]['I']=27;          // Find 'I'
        GrammarFSA[26]['i']=27;          // Find 'i'
        GrammarFSA[26][' ']=26;          // Find '

        GrammarFSA[27]['s']=28;          // Find 's'
        GrammarFSA[27]['S']=28;          // Find 'S'
        GrammarFSA[27][' ']=27;          // Find ' '

        GrammarFSA[28]['\'']=29;         // Find '\''
        GrammarFSA[28][' ']=28;          // Find ' '

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[29][i] = 29;
        GrammarFSA[29]['\\']=30;         // Find '\'
        GrammarFSA[29]['\'']=31;          // Find ' '

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[30][i] = 29;

        GrammarFSA[31]['a']=32;         // Find 'a'
        GrammarFSA[31]['A']=32;         // Find 'A'
        GrammarFSA[31][' ']=31;         // Find ' '

        GrammarFSA[32]['s']=33;         // Find 's'
        GrammarFSA[32]['S']=33;         // Find 'S'
        GrammarFSA[32][' ']=32;         // Find ' '

        GrammarFSA[33]['u']=34;         // Find 'u'
        GrammarFSA[33]['U']=34;         // Find 'U'
        GrammarFSA[33][' ']=33;         // Find ' '

        GrammarFSA[34]['b']=35;         // Find 'b'
        GrammarFSA[34]['B']=35;         // Find 'B'
        GrammarFSA[34][' ']=34;         // Find ' '

        GrammarFSA[35]['s']=36;         // Find 's'
        GrammarFSA[35]['S']=36;         // Find 'S'
        GrammarFSA[35][' ']=35;         // Find ' '

        GrammarFSA[36]['e']=37;         // Find 'e'
        GrammarFSA[36]['E']=37;         // Find 'E'
        GrammarFSA[36][' ']=36;         // Find ' '

        GrammarFSA[37]['t']=38;         // Find 't'
        GrammarFSA[37]['T']=38;         // Find 'T'
        GrammarFSA[37][' ']=37;         // Find ' '

        GrammarFSA[38]['>']=39;         // Find '>'
        GrammarFSA[38][' ']=38;         // Find ' '

        GrammarFSA[39]['=']=40;         // Find '='
        GrammarFSA[39][' ']=39;         // Find ' '

        GrammarFSA[40]['\'']=41;        // Find '''
        GrammarFSA[40][' ']=40;         // Find ' '

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[41][i] = 41;
        GrammarFSA[41]['\'']=43;        // Find '''
        GrammarFSA[41]['\\']=42;        // Find '\'

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[42][i] = 41;

        GrammarFSA[43]['-']=44;         // Find '-'
```

```
        GrammarFSA[43][' ']=43;          // Find ' '

        GrammarFSA[44]['>']=45;          // Find '>'
        GrammarFSA[44][' ']=44;          // Find ' '

        GrammarFSA[45]['{']=46;          // Find '{'
        GrammarFSA[45][' ']=45;          // Find ' '

        GrammarFSA[46]['\'']=47;         // Find '''
        GrammarFSA[46][' ']=46;          // Find ' '

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[47][i] = 47;
        GrammarFSA[47]['\\']=48;         // Find '\'
        GrammarFSA[47]['\'']=49;         // Find '''

        for ( i = 0; i < GRAMMARFSAMAXY; i++ ) GrammarFSA[48][i] = 47;

        GrammarFSA[49]['}']=50;          // Find '}'
        GrammarFSA[49]['|']=46;          // Find '
        GrammarFSA[49][' ']=49;          // Find '

        GrammarFSA[50][' ']=50;          // Find ' '
}

Grammar::~Grammar()
{

}

//
// Is the character string a sentence
//
int
Grammar::isSentence(CharString input)
{
        int i;
        CharString NonTerminal;

        //
        // Look at each terminal character.
        //
        for ( i = 0; i < NumNT; i++)
        {
                //
                // Was s non-terminal character?
                //
                NonTerminal = NT.Get(i);
                if ( input.Contains(NonTerminal) != -1 ) return false;
        }

        return true;

}


//
// scan the production of the grammar
//
int
Grammar::ScanProduction(CharString &input,
                        CharString &output,
                        const int RuleNum,
                        const int ProductionNum)

{


  int ret = -1;
  CharString temp;
  int i;
  int GuardLength;


  GuardStruct *p = (GuardStruct *) NULL;

  //
  // Validate the Rule Number
```

```
  //
  if ( RuleNum < 0 ||
          RuleNum > NumProductions ) return ret;

  //
  // Check to see of the production number of within bounds
  //
  if ( ProductionNum < 0 ||
          ProductionNum >= NumGuardProductions(RuleNum) ) return ret;


  //
  // Retrive the rule
  //
  p = GuardedCommands.Get(RuleNum);

  //
  // Check to see where the guard is contained in the current derivation.
  //
  ret = input.Contains(p->Guard);
  if ( ret == -1 ) return ret;

  //cout << "scan found character @ " << ret << endl;


  //
  // Get the first part
  //
  output = "";
  if ( ret != 0 )
  {
         for ( i = 0; i < ret; i++ )
            output = output + input[i];
  }

  //
  // replace the characters
  //
  temp = p->Statement.Get(ProductionNum);
  output = output + temp;

  //
  // Move over the number of characters in the guard and add the end characters
  //
  GuardLength = p->Guard.length();
  if ( GuardLength + ret > input.length() ) return ret;

  for ( i = GuardLength+ret; i < input.length(); i++ )
            output = output + input[i];

  return ret;
}

//
// Return the number of productions in the rule
//
int
Grammar::NumGuardProductions(int index)
{
  int ret = 0;
  GuardStruct *p = (GuardStruct *) NULL;

  //
  // Retrieve the guarded command
  //
  p = GuardedCommands.Get(index);

  if ( p ) ret =  p->NumStatements;


  return ret;
}

//
//  Return the max parallel for this grammar.
//
int
Grammar::ScanGuardForMatch(char *Current, int index)
```

```
{
  int ret;
  CharString CSCurrent;


  GuardStruct *p = (GuardStruct *) NULL;

  //
  // Check the bounds of index
  //
  if ( index < 0 ||
          index >= NumProductions ) return false;

  p = GuardedCommands.Get(index);

  //
  // Check to see if the guard is contained in the current derivation
  //
  CSCurrent = Current;

  ret = CSCurrent.Contains(p->Guard);
  if ( ret != -1 ) return true; // yes, return  offset

  return false;   //  indicate search is done.

}

//
//  Return the max parallel for this grammar.
//
int
Grammar::MaxChildren(void)
{
  int max = 0;
  GuardStruct *p = (GuardStruct *) NULL;
  // pointer to guarded command structure
  //
  // Find the guarded command with the most parallel statements.
  //
  for ( int i = 0; i < NumProductions; i++)
  {
        p = GuardedCommands.Get(i);
        if ( p->NumStatements > max ) max = p->NumStatements;
  }

  return max;

}

//
//  Add a non-terminal string to the grammar class.
//
int
Grammar::AddNT(CharString &CS)
{
        int ErrorCode;
        if ( isNT( CS ) )
                return false;
        if ( isT( CS ) )
                return false;
        ErrorCode = NT.Put(CS, NumNT);
        assert(ErrorCode );     // terminate when error
    NumNT++;    //  Increment the number of Non-Terminals that are in this grammar.
        return true;
}

//
//  Add a Terminal string to the grammar class.
//
int
Grammar::AddT(CharString &CS)
{
        int ErrorCode;
        if ( isT(CS) )
                return false;
        if ( isNT(CS) )
                return false;
        ErrorCode = T.Put(CS, NumT);
```

```
        assert(ErrorCode );     // terminate when error
    NumT++;    //  Increment the number of Non-Terminals that are in this grammar.
        return true;
}

//
//  Is the CharString CS a member of the NT set.
//
int
Grammar::isNT(CharString &CS)
{
        int i;
        CharString *p;

        if ( !NumNT ) return false;                    // Are there any members?

        for ( i = 0; i < NumNT; i++) {       // Look through and check each member
                p = NT.Get(i);                                  // Get an entry to check
                if ( !p ) continue;                             // Valid pointer?
                if ( *p == CS  ) {
                        return true;    // The same return true;
                }
        }

        return false;                                           // Not a member
}

//
//  Is the CharString CS a member of the T set.
//
int
Grammar::isT(CharString &CS)
{
        int i;
        CharString *p;

        if ( !NumT ) return false;                     // Are there any members?

        for ( i = 0; i < NumT; i++) {        // Look through and check each member
                p = T.Get(i);                                   // Get an entry to check
                if ( !p ) continue;                             // Valid pointer?
                if ( *p == CS  ) return true;    // The same return true;
        }

        return false;                                           // Not a member
}

//
//  Add a Production Guarded command
//
int
Grammar::AddProduction(CharString &CSGuard)
{
        GuardStruct *p = new GuardStruct;                       // allocate a new
GuardStructure
        p->Guard = CSGuard;                             // Set the production guard
        p->NumStatements = 0;                           // initialize the number of guard
statements
        GuardedCommands.Put(*p, NumProductions);    // Add the new guarded command to
command list
        NumProductions++;                               // Increase the number of production
rules
        p = (GuardStruct *) NULL;                       // Null this methods pointer
        return NumProductions-1;                        // return the production rule number
}

int
Grammar::AddProduction(int GuardNum, CharString &CSGuardCmd)
{
        GuardStruct *p;

    p = GuardedCommands.Get(GuardNum);  // Get the guard structure

        //
        //  Make sure there is a statement there
        //
        if ( !p ) return false;
```

```
        p->Statement.Put(CSGuardCmd, p->NumStatements);
        p->NumStatements++;
        return true;


}

//
// Overload '<<' operator.  The '<<' operator will print the string str.
//
ostream&
operator<< (ostream &out, Grammar &g)
{

  int i, ii;
  GuardStruct *p = (GuardStruct *) NULL;
// pointer to guarded command structure


  out << "Grammar:" << endl;

  //
  // Output the Non-Terminals
  //
  out << "NT={";
  for ( i = 0; i < g.NumNT; i++)
        out << ( (i) ? "," : "" ) << *g.NT.Get(i);
  out << "}" << endl;

  //
  // Output the Non-Terminals
  //
  out << "T={";
  for ( i = 0; i < g.NumT; i++)
        out << ( (i) ? "," : "" ) << *g.T.Get(i);
  out << "}" << endl;

  //
  // Output the Productions
  //
  for ( i = 0; i < g.NumProductions; i++) {
  out << "P={";
        p = g.GuardedCommands.Get(i);
      out << "(" << p->NumStatements << ")";
        out << p->Guard
         << ((p && p->NumStatements) ? ",(" : "" );
      for ( ii = 0; ii < p->NumStatements; ii++)
           out << ( (ii) ? "," : "" ) << *(p->Statement.Get(ii));
      out << ((p && p->NumStatements) ? ")" : "" ) << "}" << endl;
  }

  return out;
}

//
// Parse the input character string
//
int
Grammar::Parse(const char *input, CharString &results)
{
        int LineSize;
        LineSize = strlen(input);
    int i, j, k;
        int state = 0;
        int Addflag1, Addflag2, Addflag3, GuardsMatch, Error, ProdNum;
    CharString NonTerm, Term, StartSym;
    CharString Guard1, Guard2, Production;
        CharString *temp;


        //
        // Loop through and parse the control card
        //
        for ( i = 0; i < LineSize; i++ ) {

                state = GrammarFSA[state][input[i]];

                switch(state) {
```

```
        //
        // Do not change the state for case 0-6
        //
        case 0:
        case 1:
        case 2:
        case 3:
        case 5:
        case 6:
break;


        //
        // Restart the start Symbol for the grammar
        //
        case 8:
        case 9:
        case 10:
        case 11:
                StartSym="";
break;

        //
        // Add the character to the start symbol
        //
        case 12:
                if ( input[i] != '\'' ) StartSym=StartSym+input[i];
                break;

        case 13:
break;

        //
        // Restart the the non-terminal character(s)
        //
        case 14:
                NonTerm = "";
break;

        //
        // Add the character to the current non-terminal character(s)
        //
        case 15:
                if ( input[i] != '\'' ) NonTerm=NonTerm+int(input[i]);
                break;

        case 16:
break;

        //
        // Add the Non-terminal character to the list of non-terminal
        // characters. If it already exists then post an error.
        //
        case 17:
                temp = new CharString;
                *temp = NonTerm;
                Error = AddNT(*temp);

                //
                // Check and see if the non-terminal character was added
                //
                if ( !Error ) {
                        results = "P3: Non-terminal alread exits: ";
                        results = results + NonTerm;
                        return false;
                }
                break;

        //
        // Restart the the terminal character(s)
        //
        case 18:
                Term = "";
break;


        //
```

```
        // Add the character to the current terminal character(s)
        //
        case 19:
                if ( input[i] != '\'' ) Term=Term+int(input[i]);
                break;

        case 20:
break;

        //
        // Add the terminal character to the list of non-terminal
        // characters. If it already exists then post an error.
        //
        case 21:
                temp = new CharString;
                *temp = Term;
                Error = AddT(*temp);

                //
                // Check and see if the terminal character was added
                //
                if ( !Error ) {
                        results = "P4: Terminal alread exits: ";
                        results = results + Term;
                        return false;
                }
                break;

        case 22:
        case 23:
        case 24:
        case 25:
        case 26:
        case 27:
                break;

        //
        // Reset the guard, flags and if the guards match flag.
        //
        case 28:
Guard1 = "";
                Addflag1 = true;
                GuardsMatch = false;
                break;

        //
        // Add the current character to the first guard in the
        // guard list control card.
        //
        case 29:
                if ( input[i] != '\'' ) Guard1=Guard1+input[i];
                break;

        case 30:
                break;

        //
        // reset the addflag1 flag.
        //
        case 31:
                Addflag1 = false;
                break;

        case 32:
        case 33:
        case 34:
        case 35:
        case 36:
        case 37:
        case 38:
        case 39:
                break;
        //
        // Reset the guard2, flags and if the guards match flag.
        //
        case 40:
Guard2 = "";
                Addflag2 = true;
```

```
                        break;

                //
                // Add the current character to the second guard in the
                // guard list control card.
                //
                case 41:
                        if ( input[i] != '\'' ) Guard2=Guard2+input[i];
                        break;

                case 42:
                        break;

                //
                // reset the addflag2 flag. Compare the two guards from the control
card
                // and make sure that they are the same. If the guards are not then
                // flag an error.
                //
                case 43:
                        Addflag2 = true;

                        //
                        // Are the two guards on the guard list the same?
                        //
                        if (  Guard1 == Guard2 )
                        {
                                GuardsMatch = true;
                                temp = new CharString;
                                *temp = Guard1;
                                ProdNum = AddProduction(*temp);
                        }
                        //
                        // Produce an error message the control card is invalid.
                        //
                        else
                        {
                                results = "P1: Guards must match in production
statements! ";
                                return false;
                        }
                        break;

                case 44:
                case 45:
                        break;

                //
                // Reset the production characters string and reset the flag.
                //
                case 46:
        Production = "";
                        Addflag3 = true;
                        break;

                //
                // Add the current character to the current production.
                //
                case 47:
                        if ( input[i] != '\'' ) Production=Production+input[i];
                        break;

                case 48:
                        break;

                //
                // if the production Addflag is true add the current production
                // and reset the add flags.
                //
                case 49:
                        if ( Addflag3 )
                        {
                                temp = new CharString;
                                *temp = Production;
                                AddProduction(ProdNum, *temp);
                        }
                        Addflag3 = false;
                        break;
```

```
                        case 50:
                                break;

                        //
                        // Build parse error message
                        //
                        case 99:
                                results = "P2: Parse failed near characters: ";
                                for ( j = i, k = 0; j < LineSize, k < 6; j++, k++)
                                        results = results + input[j];
                                return false;
                                break;

                        default:
                                assert(!"Default switch in Parse!");
                                break;

                        case 4:
                        case 7:
                                assert(!"Logic error entered dead state");
                                break;

                }



        }

        //
        // Control card parse "OK"
        //
    return true;
}
```

MainFrm.cpp / MainFrm.h

The main frame header and implementation files are used to describe the main frame in the Windows MDI application. These files are as generated by the class wizard.

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__E69D48FC_4843_11D1_A845_00C0D1095F74__INCLUDED_)
#define AFX_MAINFRM_H__E69D48FC_4843_11D1_A845_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CMDIFrameWnd
{
        DECLARE_DYNAMIC(CMainFrame)
public:
        CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
```

```
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_MAINFRM_H__E69D48FC_4843_11D1_A845_00C0D1095F74__INCLUDED_)


// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "plug.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}
```

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_wndToolBar.Create(this) ||
		!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
	{
		TRACE0("Failed to create toolbar\n");
		return -1;      // fail to create
	}

	if (!m_wndStatusBar.Create(this) ||
		!m_wndStatusBar.SetIndicators(indicators,
		  sizeof(indicators)/sizeof(UINT)))
	{
		TRACE0("Failed to create status bar\n");
		return -1;      // fail to create
	}

	// TODO: Remove this if you don't want tool tips or a resizeable toolbar
	m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
		CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

	// TODO: Delete these three lines if you don't want the toolbar to
	//  be dockable
	m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_ANY);
	DockControlBar(&m_wndToolBar);

	return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CMDIFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

ParseControls.cpp / ParseControls.h

The ParseControls header/implementation files are a MFC dialog class used to display a dialog box for controlling the derivation process. After the derivation process has ended, statistics are added to the dialog box about the ended derivation process.

```
#if !defined(AFX_PARSECONTROLS_H__683BD6D3_A0E9_11D1_8F59_00C0D1095F74__INCLUDED_)
#define AFX_PARSECONTROLS_H__683BD6D3_A0E9_11D1_8F59_00C0D1095F74__INCLUDED_
```

```
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ParseControls.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CParseControls dialog

class CParseControls : public CDialog
{
// Construction
public:
        CParseControls(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CParseControls)
        enum { IDD = IDD_PARSE_CONTROLS };
        int             m_max_threads;
        int             m_total_threads;
        int             m_run_time;
        CString m_start_from;
        CString m_search_for;
        CString m_sentence_found;
        int             m_num_nodes;
        int             m_num_sentential;
        int             m_num_sentences;
        int             m_num_terminated;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CParseControls)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:


        // Generated message map functions
        //{{AFX_MSG(CParseControls)
        virtual void OnOK();
        afx_msg void OnStartProcessing();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PARSECONTROLS_H__683BD6D3_A0E9_11D1_8F59_00C0D1095F74__INCLUDED_)


// ParseControls.cpp : implementation file
//

#include "stdafx.h"
#include <afxcview.h>
#include "plug.h"
#include "ParseControls.h"
#include "List.h"
#include "CharString.h"
#include "grammar.h"
#include "SententialForm.h"
#include "Derivation.h"
#include "plugParallel.h"
#include "PlugProcessOutDoc.h"
#include "PlugTreeViewDoc.h"
#include "PlugTreeView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

enum { stop_found, stop_time, stop_iterations };
/////////////////////////////////////////////////////////////////////////////
// CParseControls dialog


CParseControls::CParseControls(CWnd* pParent /*=NULL*/)
        : CDialog(CParseControls::IDD, pParent)
{
        //{{AFX_DATA_INIT(CParseControls)
        m_max_threads = 0;
        m_total_threads = 0;
        m_run_time = 0;
        m_start_from = _T("");
        m_search_for = _T("");
        m_sentence_found = _T("");
        m_num_nodes = 0;
        m_num_sentential = 0;
        m_num_sentences = 0;
        m_num_terminated = 0;
        //}}AFX_DATA_INIT
}


void CParseControls::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CParseControls)
        DDX_Text(pDX, IDC_MAX_THREADS, m_max_threads);
        DDX_Text(pDX, IDC_TOTAL_THREADS, m_total_threads);
        DDX_Text(pDX, IDC_RUN_TIME, m_run_time);
        DDX_Text(pDX, IDC_START_FROM, m_start_from);
        DDX_Text(pDX, IDC_SEARCH_FOR, m_search_for);
        DDX_Text(pDX, IDC_SENTENCE_FOUND, m_sentence_found);
        DDX_Text(pDX, IDC_NUM_NODES, m_num_nodes);
        DDX_Text(pDX, IDC_NUM_SENTENTIAL, m_num_sentential);
        DDX_Text(pDX, IDC_NUM_SENTENCES, m_num_sentences);
        DDX_Text(pDX, IDC_NUM_TERMINATED, m_num_terminated);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CParseControls, CDialog)
        //{{AFX_MSG_MAP(CParseControls)
        ON_BN_CLICKED(IDC_START_PROCESSING, OnStartProcessing)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CParseControls message handlers


void CParseControls::OnOK()
{

        CDialog::OnOK();
}

//
// Reference the data in the application
//
extern Grammar *g;
extern Derivation *Root;
extern SententialForm Root_SF;
extern char *Reverse;

//
// The section invokes the derivation processing
//
void CParseControls::OnStartProcessing()
{
    CStringArray Parse_Results;
        char *EndString;
        char *StartString;
        char *TempString;
        Statistics Stats;
```

```
        int temp;
        time_t  StartTime, StopTime;
        CWinApp *PApp;
        POSITION curTemplatePos;


        //
        // Reset the flag indicating the the tree should be shown backwards.
        //
        Reverse = NULL;

        //
        // Get a current time for use as a start time.
        //
        time(&StartTime);

        //
        // Get the data from the dialog.
        //
        UpdateData(true);

        //
        // Allocate and copy the Ending string (sentence/sentential form of the grammar)
        //
        EndString = new char[m_search_for.GetLength()+5];
        strcpy(EndString, m_search_for);

        //
        // Allocate and copy the start string (sentence/sentential form of the grammar)
        //
        StartString = new char[m_start_from.GetLength()+5];
        strcpy(StartString, m_start_from);

        //
        // If the starting string is longer than the ending string the data tree produced
        // will be shown in a parsing (backward) direction.
        //
        if ( strlen(StartString) > strlen(EndString) )  // exchange the starting/ending
starting
        {
                TempString = StartString;
                StartString = EndString;
                EndString = TempString;
                Reverse = _strdup(EndString);
        }


        //
        // Build the ROOT of the derivation tree from the starting string.
        //
    Root_SF = (CharString ) StartString;
    Root = new Derivation(Root_SF, 0);

        //
        // Go and do the processing.
        //
        PlugParallel(*g, *Root, EndString, 0, Stats);

        //
        // Collect the stop time.
        //
        time(&StopTime);

        //
        // Calculate the stats from the run.
        //
        Stats.GetStats(m_max_threads, m_total_threads, temp);
    m_sentence_found = (temp ? " Found" : " Not Found");
    m_run_time = StopTime - StartTime;

        //
        // Get the statistics from the tree.
        //
        Root->Get_Tree_Stats(m_num_nodes, m_num_sentential, m_num_sentences,
m_num_terminated);


        //
```

```
        // Update the dialog fields.
        //
        UpdateData(false);


        //
        // Create the output doc of the processing text base.
        //

        // Get the App pointer and loop thru the doc templates looking for the
        // one we want.
        PApp = AfxGetApp();
        curTemplatePos = PApp->GetFirstDocTemplatePosition();
        while(curTemplatePos != NULL)
        {
                CDocTemplate* curTemplate = PApp->GetNextDocTemplate(curTemplatePos);
                CString str;
                curTemplate->GetDocString(str, CDocTemplate::docName);
                if(str == _T("Process"))
                {
                        CDocument *NewDoc = curTemplate->OpenDocumentFile(NULL);
                        CPlugProcessOutDoc *ptrDoc;
                        ptrDoc = (CPlugProcessOutDoc*) NewDoc;
                        ptrDoc->Document_Data.RemoveAll();
                        ptrDoc->Document_Data.SetSize(0, 25);
                        //
                        // Add each derivation from to the tree to the process doc.
                        //
                        Root->Print_Tree_To_StringArray(ptrDoc->Document_Data, "");
                 break;
                }
        }


        //
        // Create the output doc of the processing
        //

        // Get the App pointer and loop thru the doc templates looking for the
        // one we want. Creating a new tree view.
        PApp = AfxGetApp();
        curTemplatePos = PApp->GetFirstDocTemplatePosition();
        while(curTemplatePos != NULL)
        {
                CDocTemplate* curTemplate = PApp->GetNextDocTemplate(curTemplatePos);
                CString str;
                curTemplate->GetDocString(str, CDocTemplate::docName);
                if(str == _T("TreeView"))
                {
                        CDocument *NewDoc = curTemplate->OpenDocumentFile(NULL);
                        CPlugTreeViewDoc *ptrDoc;
                        ptrDoc = (CPlugTreeViewDoc*) NewDoc;
                        return;
                }
        }


        //
        // delete (no MLK) non-needed data.
        //
    delete [] EndString;
    delete [] StartString;
        delete [] Root;
}
```

Plug.cpp / Plug.h

The plug (Parallel Language Generation) header/implementation file contains the MFC application definitions. These files contain the message mapping for the main routine

for the program. This class contains the routines to define the MFC views and several windows messages are handled. The messages that are handled include the file open, save, save as, print, and print setup. The MFC class wizard generated the majority of the code.

```
// plug.h : main header file for the PLUG application
//

#if !defined(AFX_PLUG_H__E69D48F8_4843_11D1_A845_00C0D1095F74__INCLUDED_)
#define AFX_PLUG_H__E69D48F8_4843_11D1_A845_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CPlugApp:
// See plug.cpp for the implementation of this class
//


class CPlugApp : public CWinApp
{
public:
        CPlugApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugApp)
        public:
        virtual BOOL InitInstance();
        virtual CDocument* OpenDocumentFile(LPCTSTR lpszFileName);
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CPlugApp)
        afx_msg void OnAppAbout();
        afx_msg void OnFileNew();
        afx_msg void OnFileOpen();
        afx_msg void OnMenuSyntaxCheck();
        afx_msg void OnUpdateMenuProcess(CCmdUI* pCmdUI);
        afx_msg void OnMenuProcess();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PLUG_H__E69D48F8_4843_11D1_A845_00C0D1095F74__INCLUDED_)


// plug.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include <afxcview.h>
#include "plug.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "PlugParseOut.h"
```

```
#include "PlugParseOutDoc.h"
#include "PlugProcessOut.h"
#include "PlugProcessOutDoc.h"
#include "PlugTreeView.h"
#include "PlugTreeViewDoc.h"
#include "plugEditDoc.h"
#include "plugEdit.h"
#include "plugparsedialog.h"
#include "ParseControls.h"

//
//  Grammar classes
//
#include "List.h"
#include "CharString.h"
#include "Grammar.h"
#include "sententialform.h"
#include "Derivation.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugApp

BEGIN_MESSAGE_MAP(CPlugApp, CWinApp)
        //{{AFX_MSG_MAP(CPlugApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        ON_COMMAND(ID_FILE_NEW, OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
        ON_COMMAND(ID_MENU_SYNTAX_CHECK, OnMenuSyntaxCheck)
        ON_UPDATE_COMMAND_UI(ID_MENU_PROCESS, OnUpdateMenuProcess)
        ON_COMMAND(ID_MENU_PROCESS, OnMenuProcess)
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugApp construction

CPlugApp::CPlugApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CPlugApp object

CPlugApp theApp;


/////////////////////////////////////////////////////////////////////////////
// The Grammar data members.
Grammar *g = NULL;
Derivation *Root = NULL;
SententialForm Root_SF;
char *Reverse = NULL;




/////////////////////////////////////////////////////////////////////////////
// CPlugApp initialization

BOOL CPlugApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
```

```
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	// Change the registry key under which our settings are stored.
	// You should modify this string to be something appropriate
	// such as the name of your company or organization.
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	LoadStdProfileSettings();  // Load standard INI file options (including MRU)

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CMultiDocTemplate* pDocTemplate;
	pDocTemplate = new CMultiDocTemplate(
		IDR_PARSETYPE,
		RUNTIME_CLASS(CPlugParseOutDoc),
		RUNTIME_CLASS(CChildFrame), // custom MDI child frame
		RUNTIME_CLASS(CPlugParseOut));
	AddDocTemplate(pDocTemplate);

	pDocTemplate = new CMultiDocTemplate(
		IDR_EDITTYPE,
		RUNTIME_CLASS(CPlugEditDoc),
		RUNTIME_CLASS(CChildFrame), // custom MDI child frame
		RUNTIME_CLASS(CPlugEdit));
	AddDocTemplate(pDocTemplate);

	pDocTemplate = new CMultiDocTemplate(
		IDR_PROCESSTYPE,
		RUNTIME_CLASS(CPlugProcessOutDoc),
		RUNTIME_CLASS(CChildFrame), // custom MDI child frame
		RUNTIME_CLASS(CPlugProcessOut));
	AddDocTemplate(pDocTemplate);

	pDocTemplate = new CMultiDocTemplate(
		IDR_TREETYPE,
		RUNTIME_CLASS(CPlugTreeViewDoc),
		RUNTIME_CLASS(CChildFrame), // custom MDI child frame
		RUNTIME_CLASS(CPlugTreeView));
	AddDocTemplate(pDocTemplate);

	// create main MDI Frame window
	CMainFrame* pMainFrame = new CMainFrame;
	if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
		return FALSE;
	m_pMainWnd = pMainFrame;

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	// The main window has been initialized, so show and update it.
	pMainFrame->ShowWindow(m_nCmdShow);
	pMainFrame->UpdateWindow();

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
```

```
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CPlugApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CPlugApp commands
void CPlugApp::OnFileOpen()
{
        // TODO: Add your command handler code here
                // TODO: Add your control notification handler code here
    static char BASED_CODE szFilter[] = "All Text Files (*.grm)|*.grm|All Files
(*.*)|*.*||";

        CFileDialog Dlg(TRUE,                          // Open File Dialog
                                NULL,                           // Default extension
                                _T("*.grm"),            // Initial filename
                                OFN_HIDEREADONLY,       // Flags
                                szFilter );                     // Filter

        if (IDOK == Dlg.DoModal())
        {
                POSITION curTemplatePos = GetFirstDocTemplatePosition();

                while(curTemplatePos != NULL)
                {
                        CDocTemplate* curTemplate = GetNextDocTemplate(curTemplatePos);
                        CString str;
                        curTemplate->GetDocString(str, CDocTemplate::docName);

                        //
                        // Open a new edit doc from the data file entered.
                        //
                        if(str == _T("Edit"))
                        {
                                curTemplate->OpenDocumentFile(Dlg.GetPathName(), TRUE);
                                return;
                        }
                }
        }
```

```
}

void CPlugApp::OnFileNew()
{
        // Stop the initial FileNew when the application is started.
        static int first_call = FALSE;

        POSITION curTemplatePos = GetFirstDocTemplatePosition();

        //
        // Do not allow the first open doc to open a document.
        //
        if ( first_call == FALSE )
        {
                first_call = TRUE;
                return;
        }

        //
        // Open a empty edit doc (grammar)
        //
        while(curTemplatePos != NULL)
        {
                CDocTemplate* curTemplate = GetNextDocTemplate(curTemplatePos);
                CString str;
                curTemplate->GetDocString(str, CDocTemplate::docName);
                if(str == _T("Edit"))
                {
                        curTemplate->OpenDocumentFile(NULL);
                        return;
                }
        }
}



CDocument* CPlugApp::OpenDocumentFile(LPCTSTR lpszFileName)
{
        // TODO: Add your specialized code here and/or call the base class

                POSITION curTemplatePos = GetFirstDocTemplatePosition();

                //
                // Open a edit doc (called by the open file) menu pick
                //
                while(curTemplatePos != NULL)
                {
                        CDocTemplate* curTemplate = GetNextDocTemplate(curTemplatePos);
                        CString str;
                        curTemplate->GetDocString(str, CDocTemplate::docName);
                        if(str == _T("Edit"))
                        {
                                return curTemplate->OpenDocumentFile(lpszFileName, TRUE);
                        }
                }

                //
                // Logic error should never get here
                //
                AfxMessageBox("Logic error in plug.cpp");
        return CWinApp::OpenDocumentFile(lpszFileName);
}

//
// Syntax check the grammar in the edit view.
//
void CPlugApp::OnMenuSyntaxCheck()
{
        // TODO: Add your command handler code here
    CStringArray Parse_Results;
        CharString Results;
        int Rc;
        int NumErrors = 0;

        //
```

```
    // Get the active view pointer. This will be used to get the
    // data from the edit view.
    //
    CMDIFrameWnd *pFrame = (CMDIFrameWnd*)AfxGetApp()->m_pMainWnd;

CMDIChildWnd *pChild = pFrame->MDIGetActive();
CEdit *pView = (CEdit *) pChild->GetActiveView();

    int i, NumLine, CurrentBufferSize, temp;
    LPBYTE lpszBuffer = NULL;

    //
    //  Get the number of lines in the edit buffer.
    //
    NumLine = pView->GetLineCount();

    //
    // Make the parse results NULL
    //
    Parse_Results.RemoveAll();
    Parse_Results.SetSize(0, 25);

    //
    // Reallocate the grammar
    //
    if ( g != NULL ) {
            delete g;
            g = NULL;
    }

    //
    //  Does the edit buffer contain data?
    //
if ( NumLine == 1 && !pView->LineLength(pView->LineIndex(NumLine))) {
            AfxMessageBox("Please try again the edit buffer is empty!");
            return;
    }

//
    // Allocate a new grammar
    //
    g = new Grammar;

    //
    // Read each line from the active edit view
    //
    CurrentBufferSize = 0;
    int index;
    for (i = 0; i < NumLine; i++) {
            index = pView->LineIndex(i);

            if ( index == -1 ) {
                    AfxMessageBox("Logic: Asked for lineindex out of range?");
                    abort();
            }

            temp = pView->LineLength(index);  // How long is the line

            //
            // Will the line fit in the current buffer?
            // realloc if necessary
            //
            if ( temp > CurrentBufferSize ) {
                    if ( !lpszBuffer ) delete [] lpszBuffer;
                    CurrentBufferSize = temp;
                    lpszBuffer = new BYTE[CurrentBufferSize+sizeof(WORD)];
            }
            ZeroMemory(lpszBuffer, CurrentBufferSize+sizeof(WORD));

            //
            // Now that the current buffer is large enough for the current
            // line in the edit view move to the the buffer.
            //
            temp = pView->GetLine(i, (char *)lpszBuffer, CurrentBufferSize);

            //
            // Check the return codes
            //
```

```
                if ( temp == 0 ) {
                        AfxMessageBox("Logic: GetLine buffer was too small?");
                        abort();
                }

        *(lpszBuffer+temp) = '\0'; // null terminate
                Parse_Results.Add((char *) lpszBuffer); // add the line to the parse
results

                //
                // Parse the current line.
                //
                Rc = g->Parse((char *) lpszBuffer, Results);

                //
                // If the line had an error (count them)
                // Add the error message to the parse output view
                //
                if ( Rc == false ) {
                        Parse_Results.Add( Results.GetChar());
                        NumErrors++;
                }
        }

        //
        // show the parse results dialog box
        //
        CPlugParseDialog Dlg;

        Dlg.m_Numlines = NumLine;
        Dlg.m_NumErrors = NumErrors;



        //
        // continue processes unless the cancel button is pressed.
        //
        if (IDOK == Dlg.DoModal())
        {
        } else {
                return;
        }

        //
        // delete the grammar if errors.
        //
    if ( NumErrors != 0 ) {
                delete g;
                g = NULL;
        }

        //
        // allocate the new document. (Parse output)
        //  After the parse output is place in the doc, then view the
        //  data.
        //
        POSITION curTemplatePos = GetFirstDocTemplatePosition();
        while(curTemplatePos != NULL)
        {
                CDocTemplate* curTemplate = GetNextDocTemplate(curTemplatePos);
                CString str;
                curTemplate->GetDocString(str, CDocTemplate::docName);
                if(str == _T("Parse"))
                {
                        CDocument *NewDoc = curTemplate->OpenDocumentFile(NULL);
                        CPlugParseOutDoc *ptrDoc;
                        ptrDoc = (CPlugParseOutDoc*) NewDoc;

                        //
                        // move the parse output to the doc.
                        //
                        ptrDoc->Document_Data.RemoveAll();
                        ptrDoc->Document_Data.SetSize(0, 25);
                        for ( i = 0; i < Parse_Results.GetSize(); i++)
                            ptrDoc->Document_Data.Add(Parse_Results[i]);
                        return;
                }
        }
```

```
}

//
// If the grammar exists (parsed ok) then enable the process menu
//
void CPlugApp::OnUpdateMenuProcess(CCmdUI* pCmdUI)
{
        // TODO: Add your command update UI handler code here
        if ( g == NULL )
                pCmdUI->Enable(FALSE);
        else
                pCmdUI->Enable(TRUE);
}

//
// Display the process dialog
//
void CPlugApp::OnMenuProcess()
{
        // TODO: Add your command handler code here
        CParseControls Dlg;

        if (IDOK == Dlg.DoModal())
        {
        } else {
                return;
        }

}
```

Plug.rc

The Plug resource file is used by the MFC application and contains the resources necessary for the MFC windows application. MFC and the developer studio generated this file.

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
```

```
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif\r\n"
    "#include ""res\\plug.rc2""  // non-Microsoft Visual C++ edited resources\r\n"
    "#include ""afxres.rc""          // Standard components\r\n"
    "#include ""afxprint.rc""       // printing/print preview resources\r\n"
    "#endif\0"
END

#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_PLUGTYPE            ICON    DISCARDABLE     "res\\plugDoc.ico"
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\plug.ico"
IDR_CONT                ICON    DISCARDABLE     "res\\cont.ico"
IDR_STOP                ICON    DISCARDABLE     "res\\stop.ico"
IDR_TERM                ICON    DISCARDABLE     "res\\term.ico"

////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//

IDR_MAINFRAME           BITMAP  MOVEABLE PURE   "res\\Toolbar.bmp"

////////////////////////////////////////////////////////////////////////////
//
// Toolbar
//

IDR_MAINFRAME TOOLBAR DISCARDABLE  16, 15
BEGIN
    BUTTON      ID_FILE_NEW
    BUTTON      ID_FILE_SAVE
    BUTTON      ID_FILE_OPEN
    SEPARATOR
    BUTTON      ID_EDIT_CUT
    SEPARATOR
    BUTTON      ID_EDIT_COPY
    BUTTON      ID_EDIT_PASTE
    SEPARATOR
    BUTTON      ID_FILE_PRINT
    BUTTON      ID_APP_ABOUT
END


////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
```

```
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM "&Save\tCtrl+s",               ID_FILE_SAVE
        MENUITEM SEPARATOR
        MENUITEM "P&rint Setup...",             ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "Recent File",                 ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About plug...",              ID_APP_ABOUT
    END
END

IDR_PLUGTYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM "&Close",                      ID_FILE_CLOSE
        MENUITEM "&Save\tCtrl+S",               ID_FILE_SAVE
        MENUITEM "Save &As...",                 ID_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\tCtrl+P",           ID_FILE_PRINT
        MENUITEM "Print Pre&view",              ID_FILE_PRINT_PREVIEW
        MENUITEM "P&rint Setup...",             ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "Recent File",                 ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",               ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",                ID_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",               ID_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",              ID_EDIT_PASTE
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Process"
    BEGIN
        MENUITEM "Syntax Check",                ID_MENU_SYNTAX_CHECK
        MENUITEM "Process",                     ID_MENU_PROCESS
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&Cascade",                    ID_WINDOW_CASCADE
        MENUITEM "&Tile",                       ID_WINDOW_TILE_HORZ
        MENUITEM "&Arrange Icons",              ID_WINDOW_ARRANGE
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About plug...",              ID_APP_ABOUT
    END
END

////////////////////////////////////////////////////////////////////////////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
```

```
    "C",            ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    "O",            ID_FILE_OPEN,           VIRTKEY, CONTROL, NOINVERT
    "P",            ID_FILE_PRINT,          VIRTKEY, CONTROL, NOINVERT
    "S",            ID_FILE_SAVE,           VIRTKEY, CONTROL, NOINVERT
    "V",            ID_EDIT_PASTE,          VIRTKEY, CONTROL, NOINVERT
    VK_BACK,        ID_EDIT_UNDO,           VIRTKEY, ALT, NOINVERT
    VK_DELETE,      ID_EDIT_CUT,            VIRTKEY, SHIFT, NOINVERT
    VK_F6,          ID_NEXT_PANE,           VIRTKEY, NOINVERT
    VK_F6,          ID_PREV_PANE,           VIRTKEY, SHIFT, NOINVERT
    VK_INSERT,      ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    VK_INSERT,      ID_EDIT_PASTE,          VIRTKEY, SHIFT, NOINVERT
    "X",            ID_EDIT_CUT,            VIRTKEY, CONTROL, NOINVERT
    "Z",            ID_EDIT_UNDO,           VIRTKEY, CONTROL, NOINVERT
END


////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 217, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About plug"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT           "Plug Version 1.0",IDC_STATIC,24,21,119,8,SS_NOPREFIX
    DEFPUSHBUTTON   "OK",IDOK,164,18,32,14,WS_GROUP
END

IDD_PARSE DIALOG DISCARDABLE  0, 0, 170, 101
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "Parse Results"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,63,65,42,14
    LTEXT           "Lines parsed",IDC_STATIC,13,14,59,13
    EDITTEXT        IDC_EDIT1,81,14,70,13,ES_AUTOHSCROLL | ES_READONLY
    EDITTEXT        IDC_EDIT3,81,38,70,13,ES_AUTOHSCROLL | ES_READONLY
    LTEXT           "Number of Errors",IDC_STATIC,13,38,59,13
END

IDD_PARSE_CONTROLS DIALOG DISCARDABLE  0, 0, 224, 286
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Parse Controls"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_START_FROM,109,14,89,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_SEARCH_FOR,109,37,89,12,ES_AUTOHSCROLL
    PUSHBUTTON      "Start",IDC_START_PROCESSING,60,252,45,17
    DEFPUSHBUTTON   "OK",IDOK,122,252,45,17
    LTEXT           "Strart From",IDC_STARTSTRING,23,18,35,8
    LTEXT           "Search For",IDC_STARTSTRING2,23,41,36,8
    EDITTEXT        IDC_MAX_THREADS,109,83,89,12,ES_AUTOHSCROLL |
                    ES_READONLY | NOT WS_TABSTOP
    EDITTEXT        IDC_TOTAL_THREADS,109,60,89,12,ES_AUTOHSCROLL |
                    ES_READONLY | NOT WS_TABSTOP
    LTEXT           "Total Threads Used",IDC_TOTAL_THREADS_LABEL,23,64,64,8
    LTEXT           "Max Concurrent Threads",IDC_MAX_THREADS_LABEL,23,87,79,
                    8
    EDITTEXT        IDC_RUN_TIME,109,106,89,12,ES_AUTOHSCROLL | ES_READONLY |
                    NOT WS_TABSTOP
    LTEXT           "Run Time ",IDC_MAX_RUN_TIME_LABEL,23,110,34,8
    EDITTEXT        IDC_SENTENCE_FOUND,109,129,89,12,ES_AUTOHSCROLL |
                    ES_READONLY | NOT WS_TABSTOP
    LTEXT           "Sentence Found",IDC_SENTENCE_FOUND_LABEL,23,133,54,8
    EDITTEXT        IDC_NUM_NODES,109,152,89,12,ES_AUTOHSCROLL | ES_READONLY |
                    NOT WS_TABSTOP
    LTEXT           "Num of Nodes",IDC_NUM_NODES_LABEL,23,156,46,8
    EDITTEXT        IDC_NUM_SENTENCES,109,175,89,12,ES_AUTOHSCROLL |
                    ES_READONLY | NOT WS_TABSTOP
    LTEXT           "Num of Sentences",IDC_NUM_SENTENCES_LABEL,23,179,60,8
    EDITTEXT        IDC_NUM_SENTENTIAL,109,198,89,12,ES_AUTOHSCROLL |
                    ES_READONLY | NOT WS_TABSTOP
    LTEXT           "Num of Sentential Forms",IDC_NUM_SENTENTIAL_LABEL,23,
                    202,78,8
    EDITTEXT        IDC_NUM_TERMINATED,109,221,89,12,ES_AUTOHSCROLL |
                    ES_READONLY | NOT WS_TABSTOP
```

```
    LTEXT           "Num Terminated",IDC_NUM_TERMINATED_LABEL,23,225,53,8
END


#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "plug MFC Application\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "plug\0"
            VALUE "LegalCopyright", "Copyright (C) 1997\0"
            VALUE "OriginalFilename", "plug.EXE\0"
            VALUE "ProductName", "plug Application\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif    // !_MAC


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 210
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END

    IDD_PARSE, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 163
        TOPMARGIN, 6
        BOTTOMMARGIN, 94
    END

    IDD_PARSE_CONTROLS, DIALOG
    BEGIN
        LEFTMARGIN, 6
        RIGHTMARGIN, 217
        TOPMARGIN, 1
        BOTTOMMARGIN, 279
    END
END
```

```
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME           "Plug - Parallel Language Generation"
    IDR_EDITTYPE            "\nEdit\nPlug\nPlug Files (*.txt)\n.txt\nPlug.Document\nPlug
Document"
    IDR_VIEWTYPE            "\nView\nPlug\nPlug Files (*.txt)\n.txt\nPlug.Document\nPlug
Document"
    IDR_PARSETYPE           "\nParse\nPlug\nPlug Files (*.txt)\n.txt\nPlug.Document\nPlug
Document"
    IDR_PROCESSTYPE         "\nProcess\nPlug\nPlug Files
(*.txt)\n.txt\nPlug.Document\nPlug Document"
    IDR_TREETYPE            "\nTreeView\nPlug\nPlug Files
(*.txt)\n.txt\nPlug.Document\nPlug Document"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_IDLEMESSAGE     "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW             "Create a new document\nNew"
    ID_FILE_OPEN            "Open an existing document\nOpen"
    ID_FILE_CLOSE           "Close the active document\nClose"
    ID_FILE_SAVE            "Save the active document\nSave"
    ID_FILE_SAVE_AS         "Save the active document with a new name\nSave As"
    ID_FILE_PAGE_SETUP      "Change the printing options\nPage Setup"
    ID_FILE_PRINT_SETUP     "Change the printer and printing options\nPrint Setup"
    ID_FILE_PRINT           "Print the active document\nPrint"
    ID_FILE_PRINT_PREVIEW   "Display full pages\nPrint Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT            "Display program information, version number and
copyright\nAbout"
    ID_APP_EXIT             "Quit the application; prompts to save documents\nExit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1       "Open this document"
    ID_FILE_MRU_FILE2       "Open this document"
    ID_FILE_MRU_FILE3       "Open this document"
    ID_FILE_MRU_FILE4       "Open this document"
    ID_FILE_MRU_FILE5       "Open this document"
    ID_FILE_MRU_FILE6       "Open this document"
    ID_FILE_MRU_FILE7       "Open this document"
    ID_FILE_MRU_FILE8       "Open this document"
    ID_FILE_MRU_FILE9       "Open this document"
    ID_FILE_MRU_FILE10      "Open this document"
    ID_FILE_MRU_FILE11      "Open this document"
    ID_FILE_MRU_FILE12      "Open this document"
    ID_FILE_MRU_FILE13      "Open this document"
    ID_FILE_MRU_FILE14      "Open this document"
    ID_FILE_MRU_FILE15      "Open this document"
    ID_FILE_MRU_FILE16      "Open this document"
END
```

```
STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE            "Switch to the next window pane\nNext Pane"
    ID_PREV_PANE            "Switch back to the previous window pane\nPrevious Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_NEW           "Open another window for the active document\nNew Window"
    ID_WINDOW_ARRANGE       "Arrange icons at the bottom of the window\nArrange Icons"
    ID_WINDOW_CASCADE       "Arrange windows so they overlap\nCascade Windows"
    ID_WINDOW_TILE_HORZ     "Arrange windows as non-overlapping tiles\nTile Windows"
    ID_WINDOW_TILE_VERT     "Arrange windows as non-overlapping tiles\nTile Windows"
    ID_WINDOW_SPLIT         "Split the active window into panes\nSplit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR           "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL       "Erase everything\nErase All"
    ID_EDIT_COPY            "Copy the selection and put it on the Clipboard\nCopy"
    ID_EDIT_CUT             "Cut the selection and put it on the Clipboard\nCut"
    ID_EDIT_FIND            "Find the specified text\nFind"
    ID_EDIT_PASTE           "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT          "Repeat the last action\nRepeat"
    ID_EDIT_REPLACE         "Replace specific text with different text\nReplace"
    ID_EDIT_SELECT_ALL      "Select the entire document\nSelect All"
    ID_EDIT_UNDO            "Undo the last action\nUndo"
    ID_EDIT_REDO            "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR         "Show or hide the toolbar\nToggle ToolBar"
    ID_VIEW_STATUS_BAR      "Show or hide the status bar\nToggle StatusBar"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE          "Change the window size"
    AFX_IDS_SCMOVE          "Change the window position"
    AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
    AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
    AFX_IDS_SCCLOSE         "Close the active window and prompts to save the documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE       "Restore the window to normal size"
    AFX_IDS_SCTASKLIST      "Activate Task List"
    AFX_IDS_MDICHILD        "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_PREVIEW_CLOSE   "Close print preview mode\nCancel Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_MENU_PROCESS         "Process options dialog"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////


#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
```

```
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif
#include "res\plug.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources
#endif
/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

PlugEdit.cpp / PlugEdit.h

This header/implementation file is the MDI view of the guarded commands being edited. The MFC class wizard generated these files.

```
#if !defined(AFX_PLUGEDIT_H__1C905282_4A04_11D1_A847_00C0D1095F74__INCLUDED_)
#define AFX_PLUGEDIT_H__1C905282_4A04_11D1_A847_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugEdit.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugEdit view

class CPlugEdit : public CEditView
{
protected:
        CPlugEdit();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CPlugEdit)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugEdit)
        protected:
        virtual void OnDraw(CDC* pDC);      // overridden to draw this view
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CPlugEdit();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{{AFX_MSG(CPlugEdit)
        afx_msg void OnUpdateMenuSyntaxCheck(CCmdUI* pCmdUI);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.
```

```
#endif // !defined(AFX_PLUGEDIT_H__1C905282_4A04_11D1_A847_00C0D1095F74__INCLUDED_)


// PlugEdit.cpp : implementation file
//

#include "stdafx.h"
#include "plug.h"
#include "PlugEdit.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugEdit

IMPLEMENT_DYNCREATE(CPlugEdit, CEditView)

CPlugEdit::CPlugEdit()
{
}

CPlugEdit::~CPlugEdit()
{
}


BEGIN_MESSAGE_MAP(CPlugEdit, CEditView)
        //{{AFX_MSG_MAP(CPlugEdit)
        ON_UPDATE_COMMAND_UI(ID_MENU_SYNTAX_CHECK, OnUpdateMenuSyntaxCheck)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugEdit drawing

void CPlugEdit::OnDraw(CDC* pDC)
{
        CDocument* pDoc = GetDocument();
        // TODO: add draw code here
}

/////////////////////////////////////////////////////////////////////////////
// CPlugEdit diagnostics

#ifdef _DEBUG
void CPlugEdit::AssertValid() const
{
        CEditView::AssertValid();
}

void CPlugEdit::Dump(CDumpContext& dc) const
{
        CEditView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPlugEdit message handlers


void CPlugEdit::OnUpdateMenuSyntaxCheck(CCmdUI* pCmdUI)
{
        // TODO: Add your command update UI handler code here
        pCmdUI->Enable(TRUE);
}
```

This header/implementation file is the MDI document for the guarded commands being edited. The MFC class wizard generated these files.

```
#if !defined(AFX_PLUGEDITDOC_H__63D524D8_5AFD_11D1_A84B_00C0D1095F74__INCLUDED_)
#define AFX_PLUGEDITDOC_H__63D524D8_5AFD_11D1_A84B_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugEditDoc.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugEditDoc document

class CPlugEditDoc : public CDocument
{
protected:
        CPlugEditDoc();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CPlugEditDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugEditDoc)
        public:
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
        protected:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CPlugEditDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{{AFX_MSG(CPlugEditDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PLUGEDITDOC_H__63D524D8_5AFD_11D1_A84B_00C0D1095F74__INCLUDED_)


// PlugEditDoc.cpp : implementation file
//

#include "stdafx.h"
#include "plug.h"
#include "PlugEditDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
/////////////////////////////////////////////////////////////////////////////
// CPlugEditDoc
//
IMPLEMENT_DYNCREATE(CPlugEditDoc, CDocument)

CPlugEditDoc::CPlugEditDoc()
{
}

BOOL CPlugEditDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        return TRUE;
}

CPlugEditDoc::~CPlugEditDoc()
{
}


BEGIN_MESSAGE_MAP(CPlugEditDoc, CDocument)
        //{{AFX_MSG_MAP(CPlugEditDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugEditDoc diagnostics

#ifdef _DEBUG
void CPlugEditDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CPlugEditDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPlugEditDoc serialization

void CPlugEditDoc::Serialize(CArchive& ar)
{
        ((CEditView*)m_viewList.GetHead())->SerializeRaw(ar);
        return;
}

/////////////////////////////////////////////////////////////////////////////
// CPlugEditDoc commands
```

PlugParallel.cpp / PlugParallel.h

The PlugParallel files are the main processing routines of the threads. This routine also builds the derviation tree during processing. Searching the guarded grammar statements is perform by this routine.

```
#ifndef PLUGPARALLEL_H
#define PLUGPARALLEL_H


#define PROCESSINGDONE  "PlugProcessingDone"
#define STATISTICS      "PlugStatistics"
#define STARTTHREAD     "PlugStartThread"
```

```
#include "PlugStatistics.h"

//
// Thread structure used for the arg to thread.
//
struct ThreadParameters {
        ThreadParameters *Next;                                  // Pointer to next thread
parameter used
                                                                                    //    for
restart.
        ThreadParameters **RetryQueue;                  // Pointer to the retry queue.
        int              DEBUG;                                 // DEBUG flag
        Grammar             *g;                                 // Pointer to the grammar
        Derivation       *Parent;                           // Pointer to the current derivation
        char             *EndString;           // End sentence (search for)
        Statistics       *PStats;                           // Pointer to the statistics
        int              RuleNum;                           // RuleNum: Guard Rulenumber
        int              GuardOffset;          // Production offset in the rulenum
        int              ParentIndex;          // Parent offset index.
};

//
// PlugParallel: prototype
//
int PlugParallel(Grammar &g,
                                Derivation &Root,
                                char *EndString,
                                int DEBUG,
                                Statistics &Stats);
#endif


//  The following needs to be commented for console application
#include "stdafx.h"

#include "plugcons_command_line.h"



#include <windows.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include <stdio.h>

#include "CharString.h"
#include "List.h"
#include "Grammar.h"
#include "SententialForm.h"
#include "Derivation.h"


#include "PlugStatistics.h"



#include "plugParallel.h"


#ifdef _AFX
#include <afxmt.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#endif

//
// Declare the thead signiture
//
UINT PlugThread(void *p);
```

```
//
// This is the main routine for the derivation process
//
int PlugParallel(Grammar &g,
                                Derivation &ROOT,
                                char *EndString,
                                int DEBUG,
                                Statistics &Stats)
{

        ThreadParameters *Params;
        ThreadParameters *RestartQueueParams= NULL;
        CharString temp;


    //
        // Create a semaphore to post when all processing has ended
        //
        CSemaphore HProcessingDone(0,1,PROCESSINGDONE, NULL);

    //
        // Create a mutex for protecting the statistics class.
        //
        CMutex HUpdateRunningNum(false, STATISTICS, NULL);

    //
        // Create a mutex for starting threads
        //
        CMutex HStartThread(false, STARTTHREAD, NULL);


        //
        // Set the parameters to the thread
        //
        Params = new (ThreadParameters);
        assert(Params != 0);
        Params->RetryQueue = &RestartQueueParams;
        Params->Next = NULL;
        Params->DEBUG = DEBUG;
        Params->g = &g;
        Params->RuleNum = -1;
        Params->GuardOffset = -1;
        Params->ParentIndex = -1;
        Params->Parent = &ROOT;
        Params->PStats = &Stats;
        Params->EndString = EndString;


        //
        // Start the threads
        //
        Stats.AddThread(1);
        AfxBeginThread( PlugThread, (void *) (Params) );

        TRACE("(%s:%d) Waiting on sem post\n",__FILE__,__LINE__);

        //
        // The following code will need a little explaination.
        //  The project was written using a dual process pentium II (23 MHZ) process
        //  machine with 64 meg of memory.  The computer was running Windows NT version
        //  4 with service pack 4.  There seems to be a limit on the number of threads
        //  that a process may run (1600).  After this application started 1600+
        //  threads the AfxBeginThread stopped starting threads.  If this application
        //  fails to start thread they are queue on a link list until they can be
        //  retried.
        //
    while ( true  )
        {
                //
                // Check for end processing.  If the processing lock is posted and
                // the restart queue is empty then exit this processing. If they are
                // threads to be restart then restart them.
                if ( HProcessingDone.Lock(1000) == 1
                        && RestartQueueParams == NULL ) break;

                TRACE("(%s:%d) Looking for threads to restart\n",__FILE__,__LINE__);
```

```
                //
                // Has processing dropped enough to retry?
                //
                while ( Stats.GetCurThread() < 500 && RestartQueueParams != NULL)
                {
                        //
                        // Yes restart the thread.
                        //
                        TRACE("(%s:%d) Processing restart Queue not
NULL\n",__FILE__,__LINE__);

                        //
                        // Get the processing lock and reset the thread.
                        //
                        HUpdateRunningNum.Lock(INFINITE);

            Stats.AddThread(1);
                        Params = RestartQueueParams;
                        RestartQueueParams = RestartQueueParams->Next;
                        AfxBeginThread( PlugThread, (void *) (Params) );

                        //
                        // release the processing lock
                        //
                        HUpdateRunningNum.Unlock();
                }
        }

        TRACE("(%s:%d) Got sem post\n",__FILE__,__LINE__);
        return true;

}

UINT PlugThread( LPVOID p)
{
    ThreadParameters *Params;
    ThreadParameters *NextParams;


        Derivation              *NewDerivation;
        SententialForm *NewSententialForm;
        CharString              NewCharString;
        CharString              temp;
        int                             i;
        int                             j;
        int             RuleNum;
        int                             Number_of_Guard_Productions;
        int             StopThread = false;

        Params = (ThreadParameters *) p;

        //
        // Open the statistics update handle
        //
        CMutex HUpdateHandle(false, STATISTICS, NULL);

        //
        // Are we starting the process?
        //
        if (Params->RuleNum == -1 &&
                Params->GuardOffset == -1 &&
         Params->ParentIndex == -1 )
        {
                NewDerivation = Params->Parent;
    }

        //
        // Not we are not at the root node.
        //
    else
        {
                //
                // Get the parent character string.
                //
                temp = Params->Parent->Get_String();
                if ( Params->g->ScanProduction(temp,                            // Current
Character String to scan
```

```
                                                                NewCharString,
// The new character string (after rule is applied)
                                                                Params->RuleNum,
// Which rule number in the grammar
                                                                Params->GuardOffset
// Which production are we scanning
                                                                ) == -1 )
        {
                //
                // Since the guard said that this character string should
                // be found in the current sentential form this return
                // code should not happen.
                //
                TRACE("(%s:%d) logic error\n",__FILE__,__LINE__);
                exit(1);
        }

        if ( Params->DEBUG ) cout << "      ScanProduction input[" << temp
                                   << "] output [" << NewCharString
                                              << "]" << endl;

        //
        // Allocate the new sentential form for the new sentiatial form
        //
        NewSententialForm = new SententialForm(NewCharString);

        //
        // If the new sentianalform is a sentence then mark it so.
        //
        if ( Params->g->isSentence(NewCharString))
        {
                NewSententialForm->isASentence(true);
                StopThread = true;
        }
        else
                NewSententialForm->isASentence(false);

        //
        // Get the current level of the old node.
        //
        j = Params->Parent->Get_Level() + 1;

        //
        // Longer than termination string?
        //
        if ( (int) NewCharString.length() >  (int) strlen(Params->EndString) )
        {
                NewSententialForm->Terminated();
                StopThread = true;
        }

        //
        // Allocate the new derivation and add it the the tree.
        //
        NewDerivation = new Derivation(*NewSententialForm, j);
        Params->Parent->Add_Child(*NewDerivation, Params->ParentIndex);

}

//
// Check and see if we have found the termination character string
//
if ( CharString(NewDerivation->Get_String())    == CharString(Params->EndString) )
{
        Params->PStats->Found();
        StopThread = true;
}


//
// Scan each rule in the grammar.  Check each sentential form to see
// if it contain the guard?
//
if ( !StopThread)  // should the processing stop on the branch?
{
        for ( RuleNum = 0;  RuleNum < Params->g->NumGuards(); RuleNum++)
        {
```

```
                        if ( !Params->g->ScanGuardForMatch(NewDerivation->Get_String(),
RuleNum))
                                continue;

                        Number_of_Guard_Productions = Params->g-
>NumGuardProductions(RuleNum);

                        //
                        // scan the rules for productions
                        //
                        temp = NewDerivation->Get_String();

                        if ( Params->DEBUG ) cout << "      ScanGuardForMatch RuleNum [" <<
RuleNum
                                                                        << "]
Number_of_Guard_Productions [" << Number_of_Guard_Productions
                                                                        << "] Current string
[" << temp << "]" << endl;

                        //
                        // Obtain the processing lock to update stats.
                        //
                        HUpdateHandle.Lock(INFINITE);

                        Params->PStats->AddThread(Number_of_Guard_Productions);

                        HUpdateHandle.Unlock();

                        if ( Params->DEBUG ) cout << "Starting " <<
Number_of_Guard_Productions << " children" << endl;

                        //
                        // start the number of threads for this guarded command.
                        //
                        NewDerivation->SetMax(Number_of_Guard_Productions);
                        for (i = 0; i< Number_of_Guard_Productions; i++) {
                                //
                                // Set the parameters to the thread
                                //
                                NextParams = new (ThreadParameters);
                                assert(NextParams != 0);
                                Params->Next = NULL;
                                NextParams->RetryQueue = Params->RetryQueue;
                                NextParams->DEBUG = Params->DEBUG;
                                NextParams->g = Params->g;
                                NextParams->RuleNum = RuleNum;
                                NextParams->GuardOffset = i;
                                NextParams->ParentIndex = i;
                                NextParams->Parent = NewDerivation;
                                NextParams->PStats = Params->PStats;
                                NextParams->EndString = Params->EndString;

                                if ( AfxBeginThread( PlugThread, (void *) (NextParams)) ==
NULL )
                                {
                                        //
                                        // The thread failed to start add it to the restart
queue
                                        //
                                        HUpdateHandle.Lock(INFINITE);
                                        Params->PStats->DelThread(1);   // correct the
number of threads running.

                                        //
                                        // Add this thread to the restart queue
                                        //
                                        NextParams->Next = *Params->RetryQueue;
                                        Params->RetryQueue[0] = NextParams;
                                        HUpdateHandle.Unlock();

                                        TRACE("(%s:%d) Bad thread start
%d\n",__FILE__,__LINE__,GetLastError());
                                }


                        }

                }
```

```
        }

        //
        // Update the number of threads
        //
        if ( Params->DEBUG ) cout << "Terminating self " << endl;

        HUpdateHandle.Lock(INFINITE);

        if ( (Params->PStats->DelThread(1)) == 0  )
        {
                TRACE("(%s:%d) Posting Done\n",__FILE__,__LINE__);
                //
                // No more threads post main thread to terminate
                //
         CSemaphore HDoneHandle(0,1,PROCESSINGDONE, NULL);
                HDoneHandle.Unlock(1);

        }
        HUpdateHandle.Unlock();

        delete Params;

   AfxEndThread(0);
   return 0;
}
```

PlugParseDialog.cpp / PlugParseDialog.h

This is the dialog that is displayed to show the results of parsing the guarded grammar statements and the grammar definition used by this project. The MFC class wizard generated the files.

```
#if !defined(AFX_PLUGPARSEDIALOG_H__59CD7091_96B1_11D1_A872_00C0D1095F74__INCLUDED_)
#define AFX_PLUGPARSEDIALOG_H__59CD7091_96B1_11D1_A872_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugParseDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugParseDialog dialog

class CPlugParseDialog : public CDialog
{
// Construction
public:
        CPlugParseDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CPlugParseDialog)
        enum { IDD = IDD_PARSE };
        int             m_Numlines;
        int             m_NumErrors;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugParseDialog)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
```

```
        // Generated message map functions
        //{{AFX_MSG(CPlugParseDialog)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PLUGPARSEDIALOG_H__59CD7091_96B1_11D1_A872_00C0D1095F74__INCLUDED_)


// PlugParseDialog.cpp : implementation file
//

#include "stdafx.h"
#include "plug.h"
#include "PlugParseDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugParseDialog dialog


CPlugParseDialog::CPlugParseDialog(CWnd* pParent /*=NULL*/)
        : CDialog(CPlugParseDialog::IDD, pParent)
{
        //{{AFX_DATA_INIT(CPlugParseDialog)
        m_Numlines = 0;
        m_NumErrors = 0;
        //}}AFX_DATA_INIT

}


void CPlugParseDialog::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CPlugParseDialog)
        DDX_Text(pDX, IDC_EDIT1, m_Numlines);
        DDX_Text(pDX, IDC_EDIT3, m_NumErrors);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPlugParseDialog, CDialog)
        //{{AFX_MSG_MAP(CPlugParseDialog)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugParseDialog message handlers


void CPlugParseDialog::OnOK()
{

        CDialog::OnOK();
}
```

PlugParseOut.cpp / PlugParseOut.h

These header/implementation files are the MDI view of the parse results. The MFC

```
class wizard generated the files.



#if !defined(AFX_PLUGPARSEOUT_H__57192EBE_90EB_11D1_A86A_00C0D1095F74__INCLUDED_)
#define AFX_PLUGPARSEOUT_H__57192EBE_90EB_11D1_A86A_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugParseOut.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugParseOut view

class CPlugParseOut : public CScrollView
{
protected:
        CPlugParseOut();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CPlugParseOut)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugParseOut)
        protected:
        virtual void OnDraw(CDC* pDC);      // overridden to draw this view
        virtual void OnInitialUpdate();     // first time after construct
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CPlugParseOut();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
        //{{AFX_MSG(CPlugParseOut)
        afx_msg void OnUpdateMenuSyntaxCheck(CCmdUI* pCmdUI);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PLUGPARSEOUT_H__57192EBE_90EB_11D1_A86A_00C0D1095F74__INCLUDED_)


// PlugProcessOut.cpp : implementation file
//

#include "stdafx.h"
#include "plug.h"
#include "PlugProcessOut.h"
#include "PlugProcessOutDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut
```

```
IMPLEMENT_DYNCREATE(CPlugProcessOut, CScrollView)

CPlugProcessOut::CPlugProcessOut()
{
}

CPlugProcessOut::~CPlugProcessOut()
{
}


BEGIN_MESSAGE_MAP(CPlugProcessOut, CScrollView)
        //{{AFX_MSG_MAP(CPlugProcessOut)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut drawing

void CPlugProcessOut::OnInitialUpdate()
{

        CScrollView::OnInitialUpdate();

        CScrollView::OnInitialUpdate();
        CSize sizeTotal(11520, 15120);
        CSize sizePage(sizeTotal.cx / 2,
                                sizeTotal.cy / 2);
        CSize sizeLine(sizeTotal.cx / 100,
                                sizeTotal.cy / 100);
        SetScrollSizes(MM_TEXT, sizeTotal, sizePage, sizeLine);

}

void CPlugProcessOut::OnDraw(CDC* pDC)
{
        //CDocument* pDoc = GetDocument();
    // TODO: add draw code here
        CPlugProcessOutDoc* pDoc;
        pDoc = (CPlugProcessOutDoc*) GetDocument();
        CRect rc;
        int i = 0;
        int temp = 0;
        CString Temp_str;


        GetClientRect(&rc);

        ASSERT_VALID(pDoc);

        TEXTMETRIC tm;


        //
        // Get the font metrics
        //
        pDC->GetTextMetrics(&tm);


        //
        // Place all record on the screen.
        //    need to calculate the location of each line.
        //
        for ( i = 0; i < pDoc->Document_Data.GetSize(); i++)
        {
           temp = i * (tm.tmHeight+tm.tmExternalLeading);
       pDC->TextOut( 0, temp, pDoc->Document_Data[i]);
        }


}

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut diagnostics

#ifdef _DEBUG
```

```
void CPlugProcessOut::AssertValid() const
{
        CScrollView::AssertValid();
}

void CPlugProcessOut::Dump(CDumpContext& dc) const
{
        CScrollView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut message handlers
```

PlugParseOutDoc.cpp / PlugParseOutDoc.h

These header/implementation files are the MDI document containing the parse results. The MFC class wizard generated the files.

```
#if !defined(AFX_PLUGPROCESSOUTDOC_H__6C48DBDF_C18D_11D1_8F88_00C0D1095F74__INCLUDED_)
#define AFX_PLUGPROCESSOUTDOC_H__6C48DBDF_C18D_11D1_8F88_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugProcessOutDoc.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc document

class CPlugProcessOutDoc : public CDocument
{
protected:
        CPlugProcessOutDoc();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CPlugProcessOutDoc)

// Attributes
public:
    CStringArray Document_Data;

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugProcessOutDoc)
        public:
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
        protected:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CPlugProcessOutDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{{AFX_MSG(CPlugProcessOutDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_PLUGPROCESSOUTDOC_H__6C48DBDF_C18D_11D1_8F88_00C0D1095F74__INCLUDED_)


// PlugProcessOutDoc.cpp : implementation file
//

#include "stdafx.h"
#include "plug.h"
#include "PlugProcessOutDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc

IMPLEMENT_DYNCREATE(CPlugProcessOutDoc, CDocument)

CPlugProcessOutDoc::CPlugProcessOutDoc()
{
}

BOOL CPlugProcessOutDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        return TRUE;
}

CPlugProcessOutDoc::~CPlugProcessOutDoc()
{
}


BEGIN_MESSAGE_MAP(CPlugProcessOutDoc, CDocument)
        //{{AFX_MSG_MAP(CPlugProcessOutDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc diagnostics

#ifdef _DEBUG
void CPlugProcessOutDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CPlugProcessOutDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc serialization

void CPlugProcessOutDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc commands
```

PlugProcessOut.cpp / PlugProcessOut.h

These header/implementation files are the MDI view of results of the derivation processing. The MFC class wizard generated the files.

```
#if !defined(AFX_PLUGPROCESSOUT_H__6C48DBDE_C18D_11D1_8F88_00C0D1095F74__INCLUDED_)
#define AFX_PLUGPROCESSOUT_H__6C48DBDE_C18D_11D1_8F88_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugProcessOut.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut view

class CPlugProcessOut : public CScrollView
{
protected:
        CPlugProcessOut();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CPlugProcessOut)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugProcessOut)
        protected:
        virtual void OnDraw(CDC* pDC);      // overridden to draw this view
        virtual void OnInitialUpdate();     // first time after construct
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CPlugProcessOut();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
        //{{AFX_MSG(CPlugProcessOut)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PLUGPROCESSOUT_H__6C48DBDE_C18D_11D1_8F88_00C0D1095F74__INCLUDED_)


// PlugProcessOut.cpp : implementation file
//

#include "stdafx.h"
```

```
#include "plug.h"
#include "PlugProcessOut.h"
#include "PlugProcessOutDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut

IMPLEMENT_DYNCREATE(CPlugProcessOut, CScrollView)

CPlugProcessOut::CPlugProcessOut()
{
}

CPlugProcessOut::~CPlugProcessOut()
{
}


BEGIN_MESSAGE_MAP(CPlugProcessOut, CScrollView)
        //{{AFX_MSG_MAP(CPlugProcessOut)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut drawing

void CPlugProcessOut::OnInitialUpdate()
{

        CScrollView::OnInitialUpdate();

        CScrollView::OnInitialUpdate();
        CSize sizeTotal(11520, 15120);
        CSize sizePage(sizeTotal.cx / 2,
                        sizeTotal.cy / 2);
        CSize sizeLine(sizeTotal.cx / 100,
                        sizeTotal.cy / 100);
        SetScrollSizes(MM_TEXT, sizeTotal, sizePage, sizeLine);

}

void CPlugProcessOut::OnDraw(CDC* pDC)
{
        //CDocument* pDoc = GetDocument();
    // TODO: add draw code here
        CPlugProcessOutDoc* pDoc;
        pDoc = (CPlugProcessOutDoc*) GetDocument();
        CRect rc;
        int i = 0;
        int temp = 0;
        CString Temp_str;


        GetClientRect(&rc);

        ASSERT_VALID(pDoc);

        TEXTMETRIC tm;


        //
        // Get the font metrics
        //
        pDC->GetTextMetrics(&tm);


        //
        // Place all record on the screen.
        //    need to calculate the location of each line.
        //
        for ( i = 0; i < pDoc->Document_Data.GetSize(); i++)
```

```
        {
           temp = i * (tm.tmHeight+tm.tmExternalLeading);
        pDC->TextOut( 0, temp, pDoc->Document_Data[i]);
        }


}

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut diagnostics

#ifdef _DEBUG
void CPlugProcessOut::AssertValid() const
{
        CScrollView::AssertValid();
}

void CPlugProcessOut::Dump(CDumpContext& dc) const
{
        CScrollView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOut message handlers
```

PlugProcessOutDoc.cpp / PlugProcessOutDoc.h

These header/implementation files are the MDI document containing the processing results. The MFC class wizard generated the files.

```
#if !defined(AFX_PLUGPROCESSOUTDOC_H__6C48DBDF_C18D_11D1_8F88_00C0D1095F74__INCLUDED_)
#define AFX_PLUGPROCESSOUTDOC_H__6C48DBDF_C18D_11D1_8F88_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugProcessOutDoc.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc document

class CPlugProcessOutDoc : public CDocument
{
protected:
        CPlugProcessOutDoc();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CPlugProcessOutDoc)

// Attributes
public:
    CStringArray Document_Data;

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugProcessOutDoc)
        public:
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
        protected:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CPlugProcessOutDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
```

```
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{{AFX_MSG(CPlugProcessOutDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_PLUGPROCESSOUTDOC_H__6C48DBDF_C18D_11D1_8F88_00C0D1095F74__INCLUDED_)


// PlugProcessOutDoc.cpp : implementation file
//

#include "stdafx.h"
#include "plug.h"
#include "PlugProcessOutDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc

IMPLEMENT_DYNCREATE(CPlugProcessOutDoc, CDocument)

CPlugProcessOutDoc::CPlugProcessOutDoc()
{
}

BOOL CPlugProcessOutDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        return TRUE;
}

CPlugProcessOutDoc::~CPlugProcessOutDoc()
{
}


BEGIN_MESSAGE_MAP(CPlugProcessOutDoc, CDocument)
        //{{AFX_MSG_MAP(CPlugProcessOutDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc diagnostics

#ifdef _DEBUG
void CPlugProcessOutDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CPlugProcessOutDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc serialization

void CPlugProcessOutDoc::Serialize(CArchive& ar)
```

```
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
// CPlugProcessOutDoc commands
```

PlugStatistics.cpp / PlugStatistics.h

The statistics class is used to collect basic statistics about the derivation proces. The statistics collected are related to thread usage during the derivation process. These statistics include max, total and a flag to indicate if the derivation path being search for was found.

```
#ifndef MY_STATISTICS
#define MY_STATISTICS

//
// The Statistics class is used to report some basic statistics for the processing.
// The statistics report on the max/total/curr threads used. Additionally they will
// report if the search sentence was found.
//
class Statistics {


  protected:

    int  MaxThreads;            // Max threads in use
        int  CurThreads;            // Current number of threads being run
        int  TotalThreads;          // Total number of threads started.
        int  WasFound;              // Flag to indicate that the search sentence
                                    //    derivation was found.


  public:

        //
        // Constructor/destructors for the class.
        //
    Statistics(void);
    Statistics(const Statistics &s);
    ~Statistics();

        //
        // methods to add/del the current number of threads in use.
        //
        int AddThread(int n);
        int DelThread(int n);

        //
        // Return the current number of threads in use.
        int GetCurThread(void) { return CurThreads; }

        //
        // Mark the search sentence as found.
        //
        void Found(void);

        //
        // Output the statistics.
```

```
        //
        void GetStats(int &Max, int &Total, int &Found);
        friend ostream& operator<< (ostream& out, Statistics &s);

};


#endif


//  The following needs to be commented for console application
#include "stdafx.h"

#include <iostream.h>
#include "PlugStatistics.h"

#ifdef _AFX
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#endif

//
// Set the default values for the member variables.
//
Statistics::Statistics(void)
{
        MaxThreads = CurThreads = TotalThreads = WasFound = 0;
}



//
// default destructor
//
Statistics::~Statistics()
{
}

//
// Return the statistics for the thread counts
//
void
Statistics::GetStats(int &Max, int &Total, int &Found)
{
        Max =   MaxThreads;
        Total = TotalThreads;
        Found = WasFound;
}

//
// Add one to number of threads in use at present.
//
int
Statistics::AddThread(int n)
{
        TRACE("(%s:%d) CurThreads %d + %d\n",__FILE__,__LINE__,CurThreads,n);

        CurThreads+=n;  // increment the number of threads in use.

        // Correct the Max concurrent threads if necessary
        if (MaxThreads < CurThreads)  MaxThreads = CurThreads;

        TotalThreads+=n;  // Adjust the total number of threads started.

        return CurThreads;  // return the current number of threads in use.
}

//
// Adjust the number of threads down at present.
//
int
Statistics::DelThread(int n)
{
        TRACE("(%s:%d) CurThreads %d - %d\n",__FILE__,__LINE__,CurThreads, n);
```

```
    CurThreads-=n;

        //
        // Make sure and return zero of the number of threads in use is
        // zero of less (safety check)
        //
    if (CurThreads <= 0 )
        {
         TRACE("(%s:%d) CurThreads 0\n",__FILE__,__LINE__,CurThreads, n);
               return 0;
        }

        //
        // Return the current number of threads in use.
        //
        return CurThreads;
}

//
// Set a flag that indicates that the search string (sentence) was found.
//
void
Statistics::Found(void)
{
    WasFound = 1;
}

//
// If the application is a console the overload output operator can be used.
//
ostream&
operator<< (ostream& out, Statistics &s)
{
        out << "Max current threads: " << s.MaxThreads << endl
               << "Total current threads: " << s.TotalThreads << endl
               << "Found: " << s.WasFound << endl;
        return out;
}
```

PlugTreeView.cpp / PlugTreeView.h

These header/implementation files are the MDI document containing a tree view of the processing results. The MFC class wizard generated the files.

```
#if !defined(AFX_PLUGTREEVIEW_H__2B1B6405_C4D4_11D1_8F8E_00C0D1095F74__INCLUDED_)
#define AFX_PLUGTREEVIEW_H__2B1B6405_C4D4_11D1_8F8E_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugTreeView.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugTreeView view

class CPlugTreeView : public CTreeView
{
protected:
        CPlugTreeView();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CPlugTreeView)
        CImageList *CIL;

// Attributes
public:

// Operations
public:
```

```
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugTreeView)
        public:
        virtual void OnInitialUpdate();
        protected:
        virtual void OnDraw(CDC* pDC);      // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CPlugTreeView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{{AFX_MSG(CPlugTreeView)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PLUGTREEVIEW_H__2B1B6405_C4D4_11D1_8F8E_00C0D1095F74__INCLUDED_)



// PlugTreeView.cpp : implementation file
//

#include "stdafx.h"
#include <afxcview.h>
#include "plug.h"
#include "PlugTreeView.h"
#include "PlugTreeViewDoc.h"
#include <iostream.h>
#include "CharString.h"
#include "SententialForm.h"
#include "Derivation.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugTreeView

IMPLEMENT_DYNCREATE(CPlugTreeView, CTreeView)

CPlugTreeView::CPlugTreeView()
{
}

CPlugTreeView::~CPlugTreeView()
{
        if (CIL) delete CIL;
}


BEGIN_MESSAGE_MAP(CPlugTreeView, CTreeView)
        //{{AFX_MSG_MAP(CPlugTreeView)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CPlugTreeView drawing

void CPlugTreeView::OnDraw(CDC* pDC)
{
        CDocument* pDoc = GetDocument();
        // TODO: add draw code here
}

/////////////////////////////////////////////////////////////////////////////
// CPlugTreeView diagnostics

#ifdef _DEBUG
void CPlugTreeView::AssertValid() const
{
        CTreeView::AssertValid();
}

void CPlugTreeView::Dump(CDumpContext& dc) const
{
        CTreeView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPlugTreeView message handlers

extern Derivation *Root;
extern char *Reverse;
void CPlugTreeView::OnInitialUpdate()
{
        int Cont_offset, Stop_offset, Term_offset;
        CWinApp *PApp = AfxGetApp();
        TV_INSERTSTRUCT                 tvstruct;
        HTREEITEM htreeRoot;


        //
        // Obtain the tree control pointer.
        //
        CTreeCtrl &m_ctlTreeCtrl = GetTreeCtrl();

        //
        // Do the treeview intial update.
        //
        CTreeView::OnInitialUpdate();

        //
        // Create a new image list to be used by this tree view
        //
        CIL = new CImageList();
        CIL->Create(16, 16, 0/*bMask*/, 0, 8);


        //
        // Load the ICONS into the image list
        //
        Cont_offset = CIL->Add(PApp->LoadIcon(IDR_CONT));
        Stop_offset = CIL->Add(PApp->LoadIcon(IDR_STOP));
        Term_offset = CIL->Add(PApp->LoadIcon(IDR_TERM));

        //
        // Load the ICON list into the tree view.
        //
        m_ctlTreeCtrl.SetImageList(CIL, TVSIL_NORMAL);



        //
        // Create a default tree view structure.
        //
        tvstruct.hParent = NULL;
        tvstruct.hInsertAfter = TVI_SORT;
        tvstruct.item.iImage = Cont_offset;
        tvstruct.item.iSelectedImage = Cont_offset;
        tvstruct.item.pszText = "<null>";
        tvstruct.item.mask = TVIF_IMAGE | TVIF_SELECTEDIMAGE | TVIF_TEXT;
```

```
        //
        // Call the appropriate view routines. (forward/backward)
        //
        if ( Reverse )
         Root->Print_Tree_To_Tree_Reverse(Reverse,
                                                  &m_ctlTreeCtrl,
                                                  tvstruct,
                                                  htreeRoot,
                                                  Cont_offset,
                                                  Stop_offset,
                                                  Term_offset);
        else
               Root->Print_Tree_To_Tree(&m_ctlTreeCtrl,
                                                  tvstruct,
                                                  htreeRoot,
                                                  Cont_offset,
                                                  Stop_offset,
                                                  Term_offset);

}

BOOL CPlugTreeView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Add your specialized code here and/or call the base class

        cs.style |= TVS_HASLINES | TVS_LINESATROOT; // set the style of the tree

        return CTreeView::PreCreateWindow(cs);
}
```

PlugTreeViewDoc.cpp / PlugTreeViewDoc.h

These header/implementation files are the MDI document containing a tree view document for the processing processing results. The MFC class wizard generates the files.

```
#if !defined(AFX_PLUGTREEVIEWDOC_H__2B1B6406_C4D4_11D1_8F8E_00C0D1095F74__INCLUDED_)
#define AFX_PLUGTREEVIEWDOC_H__2B1B6406_C4D4_11D1_8F8E_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlugTreeViewDoc.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPlugTreeViewDoc document

class CPlugTreeViewDoc : public CDocument
{
protected:
        CPlugTreeViewDoc();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CPlugTreeViewDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPlugTreeViewDoc)
        public:
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
        protected:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL
```

```
// Implementation
public:
        virtual ~CPlugTreeViewDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{{AFX_MSG(CPlugTreeViewDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PLUGTREEVIEWDOC_H__2B1B6406_C4D4_11D1_8F8E_00C0D1095F74__INCLUDED_)


// PlugTreeViewDoc.cpp : implementation file
//

#include "stdafx.h"
#include "plug.h"
#include "PlugTreeViewDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlugTreeViewDoc

IMPLEMENT_DYNCREATE(CPlugTreeViewDoc, CDocument)

CPlugTreeViewDoc::CPlugTreeViewDoc()
{
}

BOOL CPlugTreeViewDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        return TRUE;
}

CPlugTreeViewDoc::~CPlugTreeViewDoc()
{
}


BEGIN_MESSAGE_MAP(CPlugTreeViewDoc, CDocument)
        //{{AFX_MSG_MAP(CPlugTreeViewDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlugTreeViewDoc diagnostics

#ifdef _DEBUG
void CPlugTreeViewDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CPlugTreeViewDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CPlugTreeViewDoc serialization

void CPlugTreeViewDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
// CPlugTreeViewDoc commands
```

SententialForm.cpp / SententialForm.h

The SententialForm class is an extension of the CharString class. This class is used to hold a sentential form of a grammar. A sentential form is included as a member instance of each instance of the derivation class.

```
#ifndef SENTENTIALFORMCLASS
#define SENTENTIALFORMCLASS

#if !defined(true)
#define true 1
#endif

#if !defined(false)
#define false 0
#endif

//
// The sententialform is a class that is used to desribe a sentential form of a
// grammar. Many of the methods of this class are used to detmerine the status of
// this sentential form of the grammar.
//
class SententialForm : public CharString
{

protected:
        int Sentence;                           // flag indicating that this is a sentence
        int LengthTerminated;          // flag indicating that the derivation process
                                                                // was terminated for this
sentential form.

public:
        //
        // constructor/destructors for this class.
        //
        SententialForm(int n=0);
    SententialForm(const char *s);
    SententialForm(const CharString &s);
        ~SententialForm();

        //
        // Methods used to set/get the sentence flag of this sentential form
        //
        int isASentence(void) { return Sentence; };
        void isASentence(int i) { Sentence = i; return; }

        //
        // Methods used to set/get the terminated flag of this sentential form
        //
        int WasTerminated(void) { return LengthTerminated; }
        void Terminated(void) { LengthTerminated = true; return; }
```

```
        //
        // Used to overload the '=' operator for assigning
        //
        SententialForm& operator= (const CharString &s) { CharString::operator=(s); return
*this; }

        //
        // overload the output operator
        //
        friend ostream& operator<< (ostream& out, SententialForm &s);

};

#endif


// SententialForm.cpp: implementation of the SententialForm class.
//
//////////////////////////////////////////////////////////////////////
//  The following needs to be commented for console application
#include "stdafx.h"

#if !defined(true)
#define true 1
#endif

#if !defined(false)
#define false 0
#endif

#include <string.h>
#include <iostream.h>
#include <iomanip.h>

#include "CharString.h"
#include "SententialForm.h"

#ifdef _AFX
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#endif


//////////////////////////////////////////////////////////////////////
// Construction/Destruction (call the base class constructors)
//////////////////////////////////////////////////////////////////////

SententialForm::SententialForm(int n) : CharString(n)
{
  Sentence = LengthTerminated = false;
}
SententialForm::SententialForm(const char *s) : CharString(s)
{
  Sentence = LengthTerminated = false;
}
SententialForm::SententialForm(const CharString &s) : CharString(s)
{
  Sentence = LengthTerminated = false;
}

SententialForm::~SententialForm()
{

}

//
// Overload '<<' operator.  The '<<' operator will print the string str.
//
ostream&
operator<< (ostream &out, SententialForm &s)
{
        out << (CharString) s;
    return out;
```

```
}
```

StdAfx.cpp / StdAfx.h

These header/implementation files are the generated by class wizard for MFC application.

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__E69D48FA_4843_11D1_A845_00C0D1095F74__INCLUDED_)
#define AFX_STDAFX_H__E69D48FA_4843_11D1_A845_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                     // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__E69D48FA_4843_11D1_A845_00C0D1095F74__INCLUDED_)


// stdafx.cpp : source file that includes just the standard includes
//      plug.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

List.h

This List template is used to generate a dynamically sizing array. This template class is used to implement dynamic sized arrays for the derivations, and sententialforms classes. Operations include methods to add, and retrieve members from the array. When an add to the listarray exceeds the current max, the array is resized to allow for additional members.

```
// ListArray.h: interface for the ListArray class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_ListArray_H__D9C976F7_226D_11D1_A80D_00C0D1095F74__INCLUDED_)
#define AFX_ListArray_H__D9C976F7_226D_11D1_A80D_00C0D1095F74__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#include <iostream.h>
```

```
#include <iomanip.h>
#include <assert.h>

#ifndef true
#define true 1
#endif
#ifndef false
#define false 0
#endif


#define LIST_ARRAY_INCREASE_BY 10
template <class T> class ListArray
{
protected:
        T    **TheListArray;     // pointer to the list
        int  MaxEntries;         // number of entries in the list (max)

        int Expand(int i);       // expand the list (array)

public:

        //
        // Constructors
        //
        ListArray(void);
        ~ListArray(void);

        //
        // List array operators
        //
        int Put(T &l, int offset);  // Put an entry
        T*  Get(int offset);        // Get an entry
        int SetMax(int n);          // Set the max entries
        void print();               // print an list array
};

////////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////////

//
// Print the list array to stdout
//
template <class T> void ListArray<T>::print()
{
        int i;
    cout << "----List Array= ";
        for ( i = 0; i < MaxEntries; i++)
                cout << " (" << i << "," << TheListArray[i] << ")";
        cout << endl;
        return;
}

//
// Constructor for the listarray.
//
template <class T> ListArray<T>::ListArray(void)
{
        int i;
        MaxEntries = 1;                          // set the max entries
        TheListArray = new (T *[MaxEntries]);    // Allocate the list array
        assert(TheListArray);                    // did it allocate

        for ( i = 0; i < MaxEntries; i++)        // Make sure each is NULL
                TheListArray[i] = (T *) NULL;
}


//
// destructor for the listarray.
//
template <class T> ListArray<T>::~ListArray(void)
{
  delete [] TheListArray;
}
```

```
//
// Put an entry in the list array.
//
template <class T> int ListArray<T>::Put(T &l, int offset)
{
        if ( offset < 0 ) return false;  // return a false entry of the offset is neg
    if ( offset >= MaxEntries ) Expand(offset);  // if the offset is larger than

// the current max expand.
    TheListArray[offset] = &l; // add the entry
        return true;
}



//
// Put an entry in the list array.
//
template <class T> int ListArray<T>::SetMax(int offset)
{
    Expand(offset);   // Set the max
    return true;
}


//
// Get an entry from the list array at index offset
//
template <class T> T* ListArray<T>::Get(int offset)
{
    if ( offset < 0 || offset >= MaxEntries ) return (T *) NULL;
        return TheListArray[offset];
}

//
// Expand the list array size.
//
template <class T> int ListArray<T>::Expand(int j)
{
    T   **Temp;
        int i;
        int NewMaxEntries = MaxEntries;

        while( (NewMaxEntries+=LIST_ARRAY_INCREASE_BY) < j );   // determine the new max

        //
        // save the old pointer and allocate the new listarray
        //
    Temp = TheListArray;
        TheListArray = new (T *[NewMaxEntries]);
        assert(TheListArray);

        //
        // Null all entries
        //
    for (i = 0; i < NewMaxEntries; i++ )
                TheListArray[i] = (T *) NULL;

        //
        // Copy all children from old to new pointer
        //
        for ( i = 0; i < MaxEntries; i++) {
                TheListArray[i] = Temp[i];
                Temp[i] = (T *) NULL;
        }
    MaxEntries = NewMaxEntries;

        //
        // Delete the old Children vector
        //
        delete [] Temp;

        return true;
}

#endif // !defined(AFX_ListArray_H__D9C976F7_226D_11D1_A80D_00C0D1095F74__INCLUDED_)
```

Resource.h

The MFC class wizard generated the file for resouce definitions.

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by plug.rc
//
#define IDD_ABOUTBOX                    100
#define IDD_PARSE                       101
#define IDR_MAINFRAME                   128
#define IDR_PLUGTYPE                    129
#define IDR_EDITTYPE                    129
#define IDR_VIEWTYPE                    130
#define IDR_PARSETYPE                   131
#define IDD_PARSE_CONTROLS              132
#define IDR_PROCESSTYPE                 132
#define IDR_TREETYPE                    133
#define IDR_CONT                        138
#define IDR_STOP                        140
#define IDR_TERM                        141
#define IDC_EDIT1                       1001
#define IDC_START_FROM                  1002
#define IDC_EDIT3                       1003
#define IDC_TOTAL_THREADS               1003
#define IDC_SEARCH_FOR                  1012
#define IDC_START_PROCESSING            1013
#define IDC_STARTSTRING                 1014
#define IDC_STARTSTRING2                1015
#define IDC_MAX_THREADS                 1016
#define IDC_TOTAL_THREADS_LABEL         1017
#define IDC_MAX_THREADS_LABEL           1018
#define IDC_RUN_TIME                    1019
#define IDC_MAX_RUN_TIME_LABEL          1020
#define IDC_SENTENCE_FOUND              1021
#define IDC_SENTENCE_FOUND_LABEL        1022
#define IDC_NUM_NODES                   1023
#define IDC_NUM_NODES_LABEL             1024
#define IDC_NUM_SENTENCES               1027
#define IDC_NUM_SENTENCES_LABEL         1028
#define IDC_NUM_SENTENTIAL              1029
#define IDC_NUM_SENTENTIAL_LABEL        1030
#define IDC_NUM_TERMINATED              1031
#define IDC_NUM_TERMINATED_LABEL        1032
#define ID_MENU_SYNTAX_CHECK            32771
#define ID_MENU_PROCESS                 32772

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        142
#define _APS_NEXT_COMMAND_VALUE         32773
#define _APS_NEXT_CONTROL_VALUE         1017
#define _APS_NEXT_SYMED_VALUE           102
#endif
#endif
```

Plug.rc2

The MFC class wizard generated the file for resouce definitions.

```
//
```

```
// PLUG.RC2 - resources Microsoft Visual C++ does not edit directly
//

#ifdef APSTUDIO_INVOKED
        #error this file is not editable by Microsoft Visual C++
#endif //APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
// Add manually edited resources here...

/////////////////////////////////////////////////////////////////////////////
```

Plug.mak

The makefile for the project.

```
# Microsoft Developer Studio Generated NMAKE File, Based on plug.dsp
!IF "$(CFG)" == ""
CFG=plug - Win32 Debug
!MESSAGE No configuration specified. Defaulting to plug - Win32 Debug.
!ENDIF

!IF "$(CFG)" != "plug - Win32 Release" && "$(CFG)" != "plug - Win32 Debug"
!MESSAGE Invalid configuration "$(CFG)" specified.
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "plug.mak" CFG="plug - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "plug - Win32 Release" (based on "Win32 (x86) Application")
!MESSAGE "plug - Win32 Debug" (based on "Win32 (x86) Application")
!MESSAGE
!ERROR An invalid configuration is specified.
!ENDIF

!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF

!IF  "$(CFG)" == "plug - Win32 Release"

OUTDIR=.\Release
INTDIR=.\Release
# Begin Custom Macros
OutDir=.\Release
# End Custom Macros

!IF "$(RECURSE)" == "0"

ALL : "$(OUTDIR)\plug.exe"

!ELSE

ALL : "$(OUTDIR)\plug.exe"

!ENDIF

CLEAN :
        -@erase "$(INTDIR)\CharString.obj"
        -@erase "$(INTDIR)\ChildFrm.obj"
        -@erase "$(INTDIR)\Derivation.obj"
        -@erase "$(INTDIR)\grammar.obj"
        -@erase "$(INTDIR)\MainFrm.obj"
        -@erase "$(INTDIR)\ParseControls.obj"
        -@erase "$(INTDIR)\plug.obj"
```

```
        -@erase "$(INTDIR)\plug.pch"
        -@erase "$(INTDIR)\plug.res"
        -@erase "$(INTDIR)\PlugEdit.obj"
        -@erase "$(INTDIR)\PlugEditDoc.obj"
        -@erase "$(INTDIR)\PlugParallel.obj"
        -@erase "$(INTDIR)\PlugParseDialog.obj"
        -@erase "$(INTDIR)\PlugParseOut.obj"
        -@erase "$(INTDIR)\PlugParseOutDoc.obj"
        -@erase "$(INTDIR)\PlugProcessOut.obj"
        -@erase "$(INTDIR)\PlugProcessOutDoc.obj"
        -@erase "$(INTDIR)\PlugStatistics.obj"
        -@erase "$(INTDIR)\PlugTreeView.obj"
        -@erase "$(INTDIR)\PlugTreeViewDoc.obj"
        -@erase "$(INTDIR)\SententialForm.obj"
        -@erase "$(INTDIR)\StdAfx.obj"
        -@erase "$(INTDIR)\vc50.idb"
        -@erase "$(OUTDIR)\plug.exe"

"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

CPP=cl.exe
CPP_PROJ=/nologo /MDd /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D\
 "_AFXDLL" /Fp"$(INTDIR)\plug.pch" /Yu"stdafx.h" /Fo"$(INTDIR)\\"\
 /Fd"$(INTDIR)\\" /FD /c
CPP_OBJS=.\Release/
CPP_SBRS=.

.c{$(CPP_OBJS)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.cpp{$(CPP_OBJS)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.cxx{$(CPP_OBJS)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.c{$(CPP_SBRS)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.cpp{$(CPP_SBRS)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.cxx{$(CPP_SBRS)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

MTL=midl.exe
MTL_PROJ=/nologo /D "NDEBUG" /mktyplib203 /o NUL /win32
RSC=rc.exe
RSC_PROJ=/l 0x409 /fo"$(INTDIR)\plug.res" /d "NDEBUG" /d "_AFXDLL"
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)\plug.bsc"
BSC32_SBRS= \

LINK32=link.exe
LINK32_FLAGS=/nologo /subsystem:windows /incremental:no\
 /pdb:"$(OUTDIR)\plug.pdb" /machine:I386 /out:"$(OUTDIR)\plug.exe"
LINK32_OBJS= \
        "$(INTDIR)\CharString.obj" \
        "$(INTDIR)\ChildFrm.obj" \
        "$(INTDIR)\Derivation.obj" \
        "$(INTDIR)\grammar.obj" \
        "$(INTDIR)\MainFrm.obj" \
        "$(INTDIR)\ParseControls.obj" \
        "$(INTDIR)\plug.obj" \
        "$(INTDIR)\plug.res" \
```

```
        "$(INTDIR)\PlugEdit.obj" \
        "$(INTDIR)\PlugEditDoc.obj" \
        "$(INTDIR)\PlugParallel.obj" \
        "$(INTDIR)\PlugParseDialog.obj" \
        "$(INTDIR)\PlugParseOut.obj" \
        "$(INTDIR)\PlugParseOutDoc.obj" \
        "$(INTDIR)\PlugProcessOut.obj" \
        "$(INTDIR)\PlugProcessOutDoc.obj" \
        "$(INTDIR)\PlugStatistics.obj" \
        "$(INTDIR)\PlugTreeView.obj" \
        "$(INTDIR)\PlugTreeViewDoc.obj" \
        "$(INTDIR)\SententialForm.obj" \
        "$(INTDIR)\StdAfx.obj"

"$(OUTDIR)\plug.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

OUTDIR=.\Debug
INTDIR=.\Debug
# Begin Custom Macros
OutDir=.\Debug
# End Custom Macros

!IF "$(RECURSE)" == "0"

ALL : "$(OUTDIR)\plug.exe" "$(OUTDIR)\plug.bsc"

!ELSE

ALL : "$(OUTDIR)\plug.exe" "$(OUTDIR)\plug.bsc"

!ENDIF

CLEAN :
        -@erase "$(INTDIR)\CharString.obj"
        -@erase "$(INTDIR)\CharString.sbr"
        -@erase "$(INTDIR)\ChildFrm.obj"
        -@erase "$(INTDIR)\ChildFrm.sbr"
        -@erase "$(INTDIR)\Derivation.obj"
        -@erase "$(INTDIR)\Derivation.sbr"
        -@erase "$(INTDIR)\grammar.obj"
        -@erase "$(INTDIR)\grammar.sbr"
        -@erase "$(INTDIR)\MainFrm.obj"
        -@erase "$(INTDIR)\MainFrm.sbr"
        -@erase "$(INTDIR)\ParseControls.obj"
        -@erase "$(INTDIR)\ParseControls.sbr"
        -@erase "$(INTDIR)\plug.obj"
        -@erase "$(INTDIR)\plug.pch"
        -@erase "$(INTDIR)\plug.res"
        -@erase "$(INTDIR)\plug.sbr"
        -@erase "$(INTDIR)\PlugEdit.obj"
        -@erase "$(INTDIR)\PlugEdit.sbr"
        -@erase "$(INTDIR)\PlugEditDoc.obj"
        -@erase "$(INTDIR)\PlugEditDoc.sbr"
        -@erase "$(INTDIR)\PlugParallel.obj"
        -@erase "$(INTDIR)\PlugParallel.sbr"
        -@erase "$(INTDIR)\PlugParseDialog.obj"
        -@erase "$(INTDIR)\PlugParseDialog.sbr"
        -@erase "$(INTDIR)\PlugParseOut.obj"
        -@erase "$(INTDIR)\PlugParseOut.sbr"
        -@erase "$(INTDIR)\PlugParseOutDoc.obj"
        -@erase "$(INTDIR)\PlugParseOutDoc.sbr"
        -@erase "$(INTDIR)\PlugProcessOut.obj"
        -@erase "$(INTDIR)\PlugProcessOut.sbr"
        -@erase "$(INTDIR)\PlugProcessOutDoc.obj"
        -@erase "$(INTDIR)\PlugProcessOutDoc.sbr"
        -@erase "$(INTDIR)\PlugStatistics.obj"
        -@erase "$(INTDIR)\PlugStatistics.sbr"
        -@erase "$(INTDIR)\PlugTreeView.obj"
        -@erase "$(INTDIR)\PlugTreeView.sbr"
        -@erase "$(INTDIR)\PlugTreeViewDoc.obj"
        -@erase "$(INTDIR)\PlugTreeViewDoc.sbr"
        -@erase "$(INTDIR)\SententialForm.obj"
        -@erase "$(INTDIR)\SententialForm.sbr"
```

```
        -@erase "$(INTDIR)\StdAfx.obj"
        -@erase "$(INTDIR)\StdAfx.sbr"
        -@erase "$(INTDIR)\vc50.idb"
        -@erase "$(INTDIR)\vc50.pdb"
        -@erase "$(OUTDIR)\plug.bsc"
        -@erase "$(OUTDIR)\plug.exe"
        -@erase "$(OUTDIR)\plug.ilk"
        -@erase "$(OUTDIR)\plug.pdb"

"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

CPP=cl.exe
CPP_PROJ=/nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS"\
 /D "_AFXDLL" /FR"$(INTDIR)\\" /Fp"$(INTDIR)\plug.pch" /Yu"stdafx.h"\
 /Fo"$(INTDIR)\\" /Fd"$(INTDIR)\\" /FD /c
CPP_OBJS=.\Debug/
CPP_SBRS=.\Debug/

.c{$(CPP_OBJS)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.cpp{$(CPP_OBJS)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.cxx{$(CPP_OBJS)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.c{$(CPP_SBRS)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.cpp{$(CPP_SBRS)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

.cxx{$(CPP_SBRS)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<

MTL=midl.exe
MTL_PROJ=/nologo /D "_DEBUG" /mktyplib203 /o NUL /win32
RSC=rc.exe
RSC_PROJ=/l 0x409 /fo"$(INTDIR)\plug.res" /d "_DEBUG" /d "_AFXDLL"
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)\plug.bsc"
BSC32_SBRS= \
        "$(INTDIR)\CharString.sbr" \
        "$(INTDIR)\ChildFrm.sbr" \
        "$(INTDIR)\Derivation.sbr" \
        "$(INTDIR)\grammar.sbr" \
        "$(INTDIR)\MainFrm.sbr" \
        "$(INTDIR)\ParseControls.sbr" \
        "$(INTDIR)\plug.sbr" \
        "$(INTDIR)\PlugEdit.sbr" \
        "$(INTDIR)\PlugEditDoc.sbr" \
        "$(INTDIR)\PlugParallel.sbr" \
        "$(INTDIR)\PlugParseDialog.sbr" \
        "$(INTDIR)\PlugParseOut.sbr" \
        "$(INTDIR)\PlugParseOutDoc.sbr" \
        "$(INTDIR)\PlugProcessOut.sbr" \
        "$(INTDIR)\PlugProcessOutDoc.sbr" \
        "$(INTDIR)\PlugStatistics.sbr" \
        "$(INTDIR)\PlugTreeView.sbr" \
        "$(INTDIR)\PlugTreeViewDoc.sbr" \
        "$(INTDIR)\SententialForm.sbr" \
        "$(INTDIR)\StdAfx.sbr"

"$(OUTDIR)\plug.bsc" : "$(OUTDIR)" $(BSC32_SBRS)
```

```
    $(BSC32) @<<
  $(BSC32_FLAGS) $(BSC32_SBRS)
<<

LINK32=link.exe
LINK32_FLAGS=/nologo /subsystem:windows /incremental:yes\
 /pdb:"$(OUTDIR)\plug.pdb" /debug /machine:I386 /out:"$(OUTDIR)\plug.exe"\
 /pdbtype:sept
LINK32_OBJS= \
        "$(INTDIR)\CharString.obj" \
        "$(INTDIR)\ChildFrm.obj" \
        "$(INTDIR)\Derivation.obj" \
        "$(INTDIR)\grammar.obj" \
        "$(INTDIR)\MainFrm.obj" \
        "$(INTDIR)\ParseControls.obj" \
        "$(INTDIR)\plug.obj" \
        "$(INTDIR)\plug.res" \
        "$(INTDIR)\PlugEdit.obj" \
        "$(INTDIR)\PlugEditDoc.obj" \
        "$(INTDIR)\PlugParallel.obj" \
        "$(INTDIR)\PlugParseDialog.obj" \
        "$(INTDIR)\PlugParseOut.obj" \
        "$(INTDIR)\PlugParseOutDoc.obj" \
        "$(INTDIR)\PlugProcessOut.obj" \
        "$(INTDIR)\PlugProcessOutDoc.obj" \
        "$(INTDIR)\PlugStatistics.obj" \
        "$(INTDIR)\PlugTreeView.obj" \
        "$(INTDIR)\PlugTreeViewDoc.obj" \
        "$(INTDIR)\SententialForm.obj" \
        "$(INTDIR)\StdAfx.obj"

"$(OUTDIR)\plug.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ENDIF


!IF "$(CFG)" == "plug - Win32 Release" || "$(CFG)" == "plug - Win32 Debug"
SOURCE=.\CharString.cpp
DEP_CPP_CHARS=\
        ".\CharString.h"\


!IF  "$(CFG)" == "plug - Win32 Release"


"$(INTDIR)\CharString.obj" : $(SOURCE) $(DEP_CPP_CHARS) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"


"$(INTDIR)\CharString.obj"     "$(INTDIR)\CharString.sbr" : $(SOURCE)\
 $(DEP_CPP_CHARS) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\ChildFrm.cpp
DEP_CPP_CHILD=\
        ".\ChildFrm.h"\
        ".\plug.h"\


!IF  "$(CFG)" == "plug - Win32 Release"


"$(INTDIR)\ChildFrm.obj" : $(SOURCE) $(DEP_CPP_CHILD) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"


"$(INTDIR)\ChildFrm.obj"       "$(INTDIR)\ChildFrm.sbr" : $(SOURCE) $(DEP_CPP_CHILD)\
```

```
 "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\Derivation.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_DERIV=\
        ".\CharString.h"\
        ".\Derivation.h"\
        ".\List.h"\
        ".\SententialForm.h"\


"$(INTDIR)\Derivation.obj" : $(SOURCE) $(DEP_CPP_DERIV) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_DERIV=\
        ".\CharString.h"\
        ".\Derivation.h"\
        ".\List.h"\
        ".\SententialForm.h"\


"$(INTDIR)\Derivation.obj"     "$(INTDIR)\Derivation.sbr" : $(SOURCE)\
 $(DEP_CPP_DERIV) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\grammar.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_GRAMM=\
        ".\CharString.h"\
        ".\Grammar.h"\
        ".\List.h"\


"$(INTDIR)\grammar.obj" : $(SOURCE) $(DEP_CPP_GRAMM) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_GRAMM=\
        ".\CharString.h"\
        ".\Grammar.h"\
        ".\List.h"\


"$(INTDIR)\grammar.obj"        "$(INTDIR)\grammar.sbr" : $(SOURCE) $(DEP_CPP_GRAMM)\
 "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\MainFrm.cpp
DEP_CPP_MAINF=\
        ".\MainFrm.h"\
        ".\plug.h"\


!IF  "$(CFG)" == "plug - Win32 Release"


"$(INTDIR)\MainFrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"
```

```
"$(INTDIR)\MainFrm.obj"        "$(INTDIR)\MainFrm.sbr" : $(SOURCE) $(DEP_CPP_MAINF)\
 "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\ParseControls.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PARSE=\
        ".\CharString.h"\
        ".\Derivation.h"\
        ".\Grammar.h"\
        ".\List.h"\
        ".\ParseControls.h"\
        ".\plug.h"\
        ".\PlugParallel.h"\
        ".\PlugProcessOutDoc.h"\
        ".\PlugStatistics.h"\
        ".\PlugTreeView.h"\
        ".\PlugTreeViewDoc.h"\
        ".\SententialForm.h"\


"$(INTDIR)\ParseControls.obj" : $(SOURCE) $(DEP_CPP_PARSE) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PARSE=\
        ".\CharString.h"\
        ".\Derivation.h"\
        ".\Grammar.h"\
        ".\List.h"\
        ".\ParseControls.h"\
        ".\plug.h"\
        ".\PlugParallel.h"\
        ".\PlugProcessOutDoc.h"\
        ".\PlugStatistics.h"\
        ".\PlugTreeView.h"\
        ".\PlugTreeViewDoc.h"\
        ".\SententialForm.h"\


"$(INTDIR)\ParseControls.obj" "$(INTDIR)\ParseControls.sbr" : $(SOURCE)\
 $(DEP_CPP_PARSE) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\plug.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUG_=\
        ".\CharString.h"\
        ".\ChildFrm.h"\
        ".\Derivation.h"\
        ".\Grammar.h"\
        ".\List.h"\
        ".\MainFrm.h"\
        ".\ParseControls.h"\
        ".\plug.h"\
        ".\PlugEdit.h"\
        ".\PlugEditDoc.h"\
        ".\PlugParseDialog.h"\
        ".\PlugParseOut.h"\
        ".\PlugParseOutDoc.h"\
        ".\PlugProcessOut.h"\
        ".\PlugProcessOutDoc.h"\
        ".\PlugTreeView.h"\
        ".\PlugTreeViewDoc.h"\
        ".\SententialForm.h"\
```

```
"$(INTDIR)\plug.obj" : $(SOURCE) $(DEP_CPP_PLUG_) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUG_=\
        ".\CharString.h"\
        ".\ChildFrm.h"\
        ".\Derivation.h"\
        ".\Grammar.h"\
        ".\List.h"\
        ".\MainFrm.h"\
        ".\ParseControls.h"\
        ".\plug.h"\
        ".\PlugEdit.h"\
        ".\PlugEditDoc.h"\
        ".\PlugParseDialog.h"\
        ".\PlugParseOut.h"\
        ".\PlugParseOutDoc.h"\
        ".\PlugProcessOut.h"\
        ".\PlugProcessOutDoc.h"\
        ".\PlugTreeView.h"\
        ".\PlugTreeViewDoc.h"\
        ".\SententialForm.h"\


"$(INTDIR)\plug.obj"   "$(INTDIR)\plug.sbr" : $(SOURCE) $(DEP_CPP_PLUG_)\
 "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\plug.rc
DEP_RSC_PLUG_R=\
        ".\res\cont.ico"\
        ".\res\plug.ico"\
        ".\res\plug.rc2"\
        ".\res\plugDoc.ico"\
        ".\res\stop.ico"\
        ".\res\term.ico"\
        ".\res\Toolbar.bmp"\


"$(INTDIR)\plug.res" : $(SOURCE) $(DEP_RSC_PLUG_R) "$(INTDIR)"
        $(RSC) $(RSC_PROJ) $(SOURCE)


SOURCE=.\PlugEdit.cpp
DEP_CPP_PLUGE=\
        ".\plug.h"\
        ".\PlugEdit.h"\


!IF  "$(CFG)" == "plug - Win32 Release"


"$(INTDIR)\PlugEdit.obj" : $(SOURCE) $(DEP_CPP_PLUGE) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"


"$(INTDIR)\PlugEdit.obj"      "$(INTDIR)\PlugEdit.sbr" : $(SOURCE) $(DEP_CPP_PLUGE)\
 "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugEditDoc.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUGED=\
        ".\plug.h"\
        ".\PlugEditDoc.h"\
```

```
"$(INTDIR)\PlugEditDoc.obj" : $(SOURCE) $(DEP_CPP_PLUGED) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUGED=\
        ".\plug.h"\
        ".\PlugEditDoc.h"\


"$(INTDIR)\PlugEditDoc.obj"    "$(INTDIR)\PlugEditDoc.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGED) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugParallel.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUGP=\
        ".\CharString.h"\
        ".\Derivation.h"\
        ".\Grammar.h"\
        ".\List.h"\
        ".\plugcons.h"\
        ".\plugcons_command_line.h"\
        ".\PlugParallel.h"\
        ".\PlugStatistics.h"\
        ".\SententialForm.h"\


"$(INTDIR)\PlugParallel.obj" : $(SOURCE) $(DEP_CPP_PLUGP) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUGP=\
        ".\CharString.h"\
        ".\Derivation.h"\
        ".\Grammar.h"\
        ".\List.h"\
        ".\plugcons.h"\
        ".\plugcons_command_line.h"\
        ".\PlugParallel.h"\
        ".\PlugStatistics.h"\
        ".\SententialForm.h"\


"$(INTDIR)\PlugParallel.obj"   "$(INTDIR)\PlugParallel.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGP) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugParseDialog.cpp
DEP_CPP_PLUGPA=\
        ".\plug.h"\
        ".\PlugParseDialog.h"\


!IF  "$(CFG)" == "plug - Win32 Release"


"$(INTDIR)\PlugParseDialog.obj" : $(SOURCE) $(DEP_CPP_PLUGPA) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"


"$(INTDIR)\PlugParseDialog.obj"       "$(INTDIR)\PlugParseDialog.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGPA) "$(INTDIR)" "$(INTDIR)\plug.pch"
```

```
!ENDIF

SOURCE=.\PlugParseOut.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUGPAR=\
        ".\plug.h"\
        ".\PlugParseOut.h"\
        ".\PlugParseOutDoc.h"\


"$(INTDIR)\PlugParseOut.obj" : $(SOURCE) $(DEP_CPP_PLUGPAR) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUGPAR=\
        ".\plug.h"\
        ".\PlugParseOut.h"\
        ".\PlugParseOutDoc.h"\


"$(INTDIR)\PlugParseOut.obj"  "$(INTDIR)\PlugParseOut.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGPAR) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugParseOutDoc.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUGPARS=\
        ".\plug.h"\
        ".\PlugParseOutDoc.h"\


"$(INTDIR)\PlugParseOutDoc.obj" : $(SOURCE) $(DEP_CPP_PLUGPARS) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUGPARS=\
        ".\plug.h"\
        ".\PlugParseOutDoc.h"\


"$(INTDIR)\PlugParseOutDoc.obj"       "$(INTDIR)\PlugParseOutDoc.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGPARS) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugProcessOut.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUGPR=\
        ".\plug.h"\
        ".\PlugProcessOut.h"\
        ".\PlugProcessOutDoc.h"\


"$(INTDIR)\PlugProcessOut.obj" : $(SOURCE) $(DEP_CPP_PLUGPR) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUGPR=\
        ".\plug.h"\
        ".\PlugProcessOut.h"\
        ".\PlugProcessOutDoc.h"\
```

```
"$(INTDIR)\PlugProcessOut.obj"        "$(INTDIR)\PlugProcessOut.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGPR) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugProcessOutDoc.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUGPRO=\
        ".\plug.h"\
        ".\PlugProcessOutDoc.h"\


"$(INTDIR)\PlugProcessOutDoc.obj" : $(SOURCE) $(DEP_CPP_PLUGPRO) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUGPRO=\
        ".\plug.h"\
        ".\PlugProcessOutDoc.h"\


"$(INTDIR)\PlugProcessOutDoc.obj"     "$(INTDIR)\PlugProcessOutDoc.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGPRO) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugStatistics.cpp
DEP_CPP_PLUGS=\
        ".\PlugStatistics.h"\


!IF  "$(CFG)" == "plug - Win32 Release"


"$(INTDIR)\PlugStatistics.obj" : $(SOURCE) $(DEP_CPP_PLUGS) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"


"$(INTDIR)\PlugStatistics.obj"        "$(INTDIR)\PlugStatistics.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGS) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugTreeView.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUGT=\
        ".\CharString.h"\
        ".\Derivation.h"\
        ".\List.h"\
        ".\plug.h"\
        ".\PlugTreeView.h"\
        ".\PlugTreeViewDoc.h"\
        ".\SententialForm.h"\


"$(INTDIR)\PlugTreeView.obj" : $(SOURCE) $(DEP_CPP_PLUGT) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUGT=\
        ".\CharString.h"\
        ".\Derivation.h"\
        ".\List.h"\
        ".\plug.h"\
```

```
        ".\PlugTreeView.h"\
        ".\PlugTreeViewDoc.h"\
        ".\SententialForm.h"\


"$(INTDIR)\PlugTreeView.obj"  "$(INTDIR)\PlugTreeView.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGT) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\PlugTreeViewDoc.cpp

!IF  "$(CFG)" == "plug - Win32 Release"

DEP_CPP_PLUGTR=\
        ".\plug.h"\
        ".\PlugTreeViewDoc.h"\


"$(INTDIR)\PlugTreeViewDoc.obj" : $(SOURCE) $(DEP_CPP_PLUGTR) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"

DEP_CPP_PLUGTR=\
        ".\plug.h"\
        ".\PlugTreeViewDoc.h"\


"$(INTDIR)\PlugTreeViewDoc.obj"       "$(INTDIR)\PlugTreeViewDoc.sbr" : $(SOURCE)\
 $(DEP_CPP_PLUGTR) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\SententialForm.cpp
DEP_CPP_SENTE=\
        ".\CharString.h"\
        ".\SententialForm.h"\


!IF  "$(CFG)" == "plug - Win32 Release"


"$(INTDIR)\SententialForm.obj" : $(SOURCE) $(DEP_CPP_SENTE) "$(INTDIR)"\
 "$(INTDIR)\plug.pch"


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"


"$(INTDIR)\SententialForm.obj"        "$(INTDIR)\SententialForm.sbr" : $(SOURCE)\
 $(DEP_CPP_SENTE) "$(INTDIR)" "$(INTDIR)\plug.pch"


!ENDIF

SOURCE=.\StdAfx.cpp
DEP_CPP_STDAF=\
        ".\StdAfx.h"\


!IF  "$(CFG)" == "plug - Win32 Release"

CPP_SWITCHES=/nologo /MDd /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D\
 "_AFXDLL" /Fp"$(INTDIR)\plug.pch" /Yc"stdafx.h" /Fo"$(INTDIR)\\"\
 /Fd"$(INTDIR)\\" /FD /c

"$(INTDIR)\StdAfx.obj" "$(INTDIR)\plug.pch" : $(SOURCE) $(DEP_CPP_STDAF)\
 "$(INTDIR)"
        $(CPP) @<<
  $(CPP_SWITCHES) $(SOURCE)
<<


!ELSEIF  "$(CFG)" == "plug - Win32 Debug"
```

```
CPP_SWITCHES=/nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D\
 "_WINDOWS" /D "_AFXDLL" /FR"$(INTDIR)\\" /Fp"$(INTDIR)\plug.pch" /Yc"stdafx.h"\
 /Fo"$(INTDIR)\\" /Fd"$(INTDIR)\\" /FD /c

"$(INTDIR)\StdAfx.obj" "$(INTDIR)\StdAfx.sbr" "$(INTDIR)\plug.pch" : $(SOURCE)\
 $(DEP_CPP_STDAF) "$(INTDIR)"
        $(CPP) @<<
  $(CPP_SWITCHES) $(SOURCE)
<<


!ENDIF


!ENDIF
```

VITA

Gregory R. Gudenburr

Candidate for the Degree of

Master of Science

Thesis: A STUDY OF FORMAL PARALLEL LANGUAGE GENERATION

Major Field: Computer Science

Biographical:

Personal Data: Born in Oklahoma City, Oklahoma, on August 26, 1958, the son of Raymond and Mary A. Gudenburr.

Education: Graduated from Bishop McGuinness High School, Oklahoma City, Oklahoma, in May 1976; attended Oklahoma State University from September 1976 until July 1980 studying Computer Science and received Bachelor of Science degree in Computer Science in July 1980; completed the requirements for the Master of Science degree in Computer Science at the Computer Science Department at Oklahoma State University in August 1998.

Experience: Worked as a Systems Analyst in the Computer Department of Conoco, in Ponca City, OK, from September 1980 until December 1996. Currently working as a Product Developer in the Research and Development Department for BMC Software, Houston, TX.