SSL:

Code Listing

Authors:

Meera Ganesan (meera.ganesan@intel.com)
Dennis Kim (dkim@harris.com)
Sandy MacDonald (sandymac@att.com)
Satheesha Rangegowda (satheesha_rangegowda_923@agilent.com)

Table of Contents

SSL.cpp 3
SSL.g 4
symbol.h 21
symbol.c 22
SSLPkg.h 25
SSLPkg.cpp 26
SSLLib.h 29
SSLLib.cpp 30
Makefile 36
README.txt 38

SSL.cpp

The main SSL program which drives compilation.

```
//
// SSL compiler program
//   authors: SSL Team
//

#include <iostream>
#include <fstream>
#include "SSLLexer.hpp"
#include "SSLParser.hpp"
#include "SSLParserTokenTypes.hpp"
#include "SSLWalker.hpp"
#include "SSLPkg.h"

char FILENAME[256];                         // global file name

void main(int argc, char * argv[])
{
                              ANTLR_USING_NAMESPACE(std)
                                          try {
   strcpy(FILENAME, argv[1]);                    // input file name
   ifstream InputFile(FILENAME, ios::in);        // input file name stream
   strcpy(strrchr(FILENAME, '.') + 1, "cpp");    // target file name
   ofstream TargetFile(FILENAME, ios::out);      // create target file
   TargetFile << "#include \"SSLLib.h\"\n\n" << "void main()\n{\n" << endl;
   TargetFile.close();                           // close target file
                                                        SSLLexer lexer(InputFile);
                                                         SSLParser parser(lexer);
                                                              parser.program();
                                            //cout << parser.getAST()->toStringList() << endl;
                                                             SSLWalker walker;
                                                    walker.program(parser.getAST());                    //
                                                     //cout << "done walking" << endl;
   TargetFile.open(FILENAME, ios::app);
   TargetFile << "\n}" << endl;
   TargetFile.close();
                                  } catch(exception& e) {
                                           cerr << "exception: " << e.what() << endl;
                                          }
}
```

SSL.g

The Antlr input which generated the Lexer, Parser, and Tree Walker.

```
//
//antlr input for SSL Compiler
//

// antlr general options
options {
        language="Cpp";                 // tell antlr to generate c++ compiler code
}


// c++ includes
{
#include <iostream>
#include <string>
}


//
//SSL Lexer Rules
// author: Sandy MacDonald
//

class SSLLexer extends Lexer;

 // antlr lexer options
 options {
        k=2;                            // lookahead 2 for WS & Comments
        charVocabulary = '\3'..'\377'; // define character set to find exclusions
        testLiterals = false;           // only test identifiers, not string values
        }


 // defines the tokens
 SEMI:      ';' ;
 RBRACE:    '{' ;
 LBRACE:    '}' ;
 EQUAL:     '=' ;
 COMMA:     ',' ;
```

```
protected LETTER: ('a'..'z'|'A'..'Z');
protected DIGIT: ('0'..'9');
ID options {testLiterals=true;}: LETTER (LETTER | DIGIT | '_')*;

INT: (DIGIT)+;


CHAR: '\''!   ( (LETTER | DIGIT | SEMI | RBRACE | LBRACE | EQUAL | COMMA)
                | ('!'|'"'|'#'|'$'|'%'|'&'|'('|')'|'*'|'+'|'-'|'.'|'/'|
                  ':'|'<'|'>'|'?'|'@'|'['|']'|'^'|'_'|'`'|'|'|'~' )
                |' '
                |'          '
                | '\"'
                | '\\'
                )
        '\''!
        ;


STRING: '"'! ( ( '"'! '"' ) |  ~( '"' ))* '"'!;

COMMENT: "/*" ( options {greedy=false;} :
            ('\r' '\n') => '\r' '\n'   {newline();}
           | '\n'                      {newline();}
           | ~( '\r' | '\n' ) )* "*/"
                {_ttype = ANTLR_USE_NAMESPACE(antlr)Token::SKIP;} ;

CPPCOMMENT : "//" ( ~('\n' | '\r' ) )* ('\n' | ('\r' '\n') )
                {_ttype = ANTLR_USE_NAMESPACE(antlr)Token::SKIP; newline();};

WS: (' ' | '\t' | '\n' {newline();} | '\r' '\n' {newline();} )
                {_ttype = ANTLR_USE_NAMESPACE(antlr)Token::SKIP;};


// END OF LEXER
```

```
//
//SSL Parser Rules
//  author: Sandy MacDonald
//

class SSLParser extends Parser;

  // antlr parser options
  options {
            k=2;                          // lookahead 2 for concatstr
            buildAST=true;                // tell antlr to build AST
            }

  // identifies additional tokens in the AST
  tokens {STRINGVALUE; CHARVALUE; INTVALUE; CONCATVALUE;}


  // defines the grammar
  program:  ((stmt1|stmt2|stmt3) SEMI!)+ EOF!;


  stmt1: ("File"^ ID EQUAL (strval|stmt2))
       |("Directory"^ ID EQUAL strval)
       |("String"^ ID EQUAL (strval|concatstr))
          ;


  stmt2: ("Find"^ (ID|strval|concatstr) "in"! (ID|strval)
                    ("using"! (("CaseOn" (COMMA!
("SubDirectoryOn"|"SubDirectoryOff"))? )
                              |("CaseOff" (COMMA! ("SubDirectoryOn"|"SubDirectoryOff"))?
)
                              |("SubDirectoryOn" (COMMA! ("CaseOn"|"CaseOff"))? )
                              |("SubDirectoryOff" (COMMA! ("CaseOn"|"CaseOff"))? )
                              )
                    )?
                    (RBRACE! ((stmt3|stmt4) SEMI!)* LBRACE!)
           )
          |("Replace"^ (ID|strval|concatstr) "in"! (ID|strval) "with"! (ID|strval|concatstr)
                    ("using"! (("CaseOn" (COMMA!
("SubDirectoryOn"|"SubDirectoryOff"))? )
                              |("CaseOff" (COMMA! ("SubDirectoryOn"|"SubDirectoryOff"))?
)
                              |("SubDirectoryOn" (COMMA! ("CaseOn"|"CaseOff"))? )
                              |("SubDirectoryOff" (COMMA! ("CaseOn"|"CaseOff"))? )
                              )
```

```
                )?
                (RBRACE! ((stmt3|stmt4) SEMI!)* LBRACE!)
        )
        ;


stmt3: ("Print"^ (ID|strval|concatstr))
    |("Output"^ "to"! (ID|strval) (ID|strval|concatstr))
        ;


stmt4: ("ReplaceLine"^ (ID|strval|concatstr))
        ;


// force identifing node in AST
concatstr: (ID|strval|charval|intval) (ID|strval|charval|intval)+
                { #concatstr  = #([CONCATVALUE, "ConcatValue"], #concatstr);}
        ;


// force identifying node in AST
strval: s:STRING      { #strval  = #([STRINGVALUE, "StringValue"], #strval);}
        ;

// force identifying node in AST
charval: c:CHAR       { #charval = #([CHARVALUE, "CharValue"], #charval);}
        ;

// force identifying node in AST
intval: i:INT  { #intval  = #([INTVALUE, "IntValue"], #intval);}
        ;


// END OF PARSER
```

```
//
// Tree Walker for Semantic Analysis
//   and Code generation
//  authors: Sateesha Rangegowda, Meera Ganesan
//


{
#include <iostream>
#include "SSLPkg.h"

extern "C" {
#include "symbol.h"
}

}

class SSLWalker extends TreeParser;

{
        protected:
                char buf[80];
}

program
        :       { Insert("LineValue", StrVar, "LineValue"); }
                ( stmt1 | stmt2 | print_stmt | output_stmt | replace_stmt)+
        ;

stmt1: { char *x2 = NULL;}
          #("File" x1:ID EQUAL (#(STRINGVALUE x2=strORid))?)
                {
                        char temp[80];
                        sprintf(temp, "%s",x1->getText().c_str());
                        if ( Lookup(temp) != NULL)
           std::cout << "Duplicate File Name ...        " << temp;

                        if (x2 == NULL)
                                Insert(temp, FileVar, "");
                        else
                                Insert(temp, FileVar, x2);
                }
         | { char *x2 = NULL;} #("Directory" x3:ID EQUAL (#(STRINGVALUE
x2=strORid))?)
                {
                        char temp[80];
```

```
                        sprintf(temp, "%s",x3->getText().c_str());
                        if ( Lookup(temp) != NULL)
            std::cout << "Duplicate Directory Name ...          " << temp;

                        if (x2 == NULL)
                                Insert(temp, DirVar, "");
                        else
                                Insert(temp, DirVar, x2);

                }

          | { char *x2 =  NULL;} #("String" x5:ID EQUAL (#(STRINGVALUE
x2=strORid))?)
                {
                        char temp[80];
                        sprintf(temp, "%s",x5->getText().c_str());
                        if ( Lookup(temp) != NULL)
            std::cout << "Duplicate String Name ...     " << temp;

                        if (x2 == NULL)
                                Insert(temp, StrVar, "");
                        else
                                Insert(temp, StrVar, x2);

                }

    | { char *x2;} #("Constant" x7:ID EQUAL (#(STRINGVALUE x2=strORid))?)
                {
                        char temp[80];
                        sprintf(temp, "%s",x7->getText().c_str());
                        if ( Lookup(temp) != NULL)
            std::cout << "Duplicate Constant Name ... " << temp;

                        if (x2 == NULL)
                                Insert(temp, Constant, "");
                        else
                                Insert(temp, Constant, x2);
                }


        ;

stmt2 { char *a1, *a2, *a3=NULL, *a4=NULL, *a5=NULL;
                bool cas = false; bool subdir = false;
                bool err = false;}
:  #("Find" (#(STRINGVALUE a1=strORid) | a1=strORid)
```

```
                    (#(STRINGVALUE a2=strORid) | a2=strORid)
                    (cas = caseOnOff)? (subdir = subDirOnOff)?
                    {

                            struct _Symbol *tst = NULL;
                            PrnRec rec;

                            initPrnRec(&rec);

                            tst = Lookup(a1);
                            if (tst != NULL)
                            {
                                    if (tst->type != StrVar)
                                    {
                                            std::cout << "invalid type ...   " <<  a1<< "
StrVar expected" << std::endl;
                                            err = true;
                                    }
                                    else
                                            sprintf(a1, "%s", tst->attrib);

                            }

                            tst = Lookup(a2);
                            if (tst != NULL )
                            {
                                    if (tst->type != FileVar)
                                    {
                                            std::cout <<  "invalid type ...  " <<  a2 << "
FileVar Expected" << std::endl;
                                            err = true;
                                    }
                                    else
                                            sprintf(a2, "%s", tst->attrib);

                            }
                    }

            (#("Print" (#(STRINGVALUE a3=strORid) | a3=strORid)) |
             #("Output" #(STRINGVALUE a4=strORid) a5=strORid))*)
                    {
                            tst = Lookup(a3);
                            if (tst != NULL)
                            {
                                    if (tst->type != StrVar)
                                    {
```

```
                        std::cout << "invalid type ...   " <<  a3<< " StrVar
expected" << std::endl;
                                err = true;
                        }
                        else
                        {
                                if(strcmp(tst->attrib, "LineValue") == 0)
                                        std::cout << "print LineValue ";

                                sprintf(a3, "%s", tst->attrib);

                                rec.LineNum = true;
                                rec.LineValue = true;
                                rec.Console = true;

                        }

                }


                tst = Lookup(a4);
                if (tst != NULL )
                {
                        if (tst->type != FileVar)
                        {
                std::cout <<  "invalid type ...  " <<  a4 << " FileVar
Expected" << std::endl;
                                err = true;
                        }
                        else
                        {
                                sprintf(a4, "%s", tst->attrib);
                        }

                }

                if (a4 != NULL && !err)
                {
                        std::cout << "Output to File " << a4 ;

                        sprintf(rec.FileStr, "%s", a4);
                        rec.File = true;
                }

                tst = Lookup(a5);
                if (tst != NULL )
```

```
                                {
                                        if (tst->type != StrVar)
                                        {
                                std::cout <<  "LineValue keyword expected" << std::endl;
                                                err = true;
                                        }
                                        else
                                        {
                                                if(strcmp(tst->attrib, "LineValue") == 0)
                                                        std::cout << "Output to  LineValue ";

                                                sprintf(a5, "%s", tst->attrib);

                                        }

                                }


                                if (!err)
                                {
                                        std::cout << "Emit Code for FindStr("
                                                << a1 << ","
                                                << a2 << ","
                                                << "case = " << cas << ","
                                                << "subDir = " << subdir << ")"
                                                << std::endl;

                                        FindStr(a1, a2, cas, subdir, &rec);

                                }

                                free (a1);
                                free(a2);
                                free (a3);
                                free(a4);
                                free(a5);
                        }
        ;



print_stmt { char *a1;}
:  #("Print" (#(STRINGVALUE a1=strORid) | a1=strORid) )
   {
       bool err = false;
                struct _Symbol *tst = NULL;
```

```
                tst = Lookup(a1);
                if (tst != NULL )
                {
                        if (tst->type != StrVar)
                        {
                                std::cout <<  "invalid type ...  " <<  a1 << " StrVar
Expected" << std::endl;
                                err = true;
                        }
                        else
                                sprintf(a1, "%s", tst->attrib);

                }

                if (a1 != NULL)
                        std::cout << "Print the String " << a1 << std::endl;


                 if (!err)
                        {
                                std::cout << "Emit Code for Print("
                                        << a1
                                        << ")"
                                        << std::endl;

                                emitCodeForPrint(a1);
                        }

                free (a1);
    };

output_stmt {char *a1, *a2;}
:  #("Output"
                (#(STRINGVALUE a1=strORid) | a1=strORid)
                (#(STRINGVALUE a2=strORid) | a2=strORid))
   {
            bool err = false;
      struct _Symbol *tst = NULL;

                tst = Lookup(a1);
                if (tst != NULL )
                {
                        if (tst->type != FileVar)
                        {
```

```
                                std::cout <<  "invalid type ...  " <<  a1 << " FileVar
Expected" << std::endl;
                                err = true;
                        }
                        else
                                sprintf(a1, "%s", tst->attrib);
                }

                if (a1 != NULL)
                        std::cout << "Output to File " << a1 << std::endl;


                tst = Lookup(a2);
                if (tst != NULL )
                {
                        if (tst->type != StrVar)
                        {
                                std::cout <<  "invalid type ...  " <<  a2 << " StrVar
Expected" << std::endl;
                                err = true;
                        }
                        else
                                sprintf(a2, "%s", tst->attrib);

                }

                if (a2 != NULL)
                        std::cout << "Output the String " << a2 << std::endl;


                if (!err)
                {
                        std::cout << "Emit Code for OutputTo("
                                << a1 << ","
                                << a2 << ")"
                                << std::endl;

                        emitCodeForOutput(a1, a2);
                }

                        free (a1);
                        free(a2);
                }
        ;

replace_stmt { char *a1, *a2, *a3, *a4=NULL, *a5=NULL,
```

```
                *a6=NULL, *a7=NULL, *a8=NULL, *a9=NULL;
                bool cas, subdir; bool err = false;}
:  #("Replace" (#(STRINGVALUE a1=strORid) | a1=strORid)
                        (#(STRINGVALUE a2=strORid) | a2=strORid)
                        (#(STRINGVALUE a3=strORid) | a3=strORid)
                        (cas = caseOnOff)? (subdir = subDirOnOff)?
                {

                        struct _Symbol *tst = NULL;
                        PrnRec rec;

                        initPrnRec(&rec);

                        tst = Lookup(a1);
                        if (tst != NULL)
                        {
                                if (tst->type != StrVar)
                                {
                                        std::cout << "invalid type ...   " <<  a1<< "
StrVar expected" << std::endl;
                                        err = true;
                                }
                                else
                                        sprintf(a1, "%s", tst->attrib);

                        }

                        tst = Lookup(a2);
                        if (tst != NULL )
                        {
                                if (tst->type != FileVar)
                                {
                                        std::cout <<  "invalid type ...  " <<  a2 << "
FileVar Expected" << std::endl;
                                        err = true;
                                }
                                else
                                        sprintf(a2, "%s", tst->attrib);

                        }


                        tst = Lookup(a3);
                        if (tst != NULL )
                        {
                                if (tst->type != StrVar)
```

```
                                        {
                                                std::cout <<  "invalid type ...  " <<  a3 << "
StrVar Expected" << std::endl;
                                                err = true;
                                        }
                                        else
                                                sprintf(a3, "%s", tst->attrib);

                                }

                        }

                (#("Print" a4=strORid) |
                 #("Output" #(STRINGVALUE a5=strORid) a6=strORid))*)
                 {
                                tst = Lookup(a4);
                                if (tst != NULL)
                                {
                                        if (tst->type != StrVar)
                                        {
                                std::cout << "invalid type ...   " <<  a4<< " StrVar
expected" << std::endl;
                                                err = true;
                                        }
                                        else
                                        {
                                                if(strcmp(tst->attrib, "LineValue") == 0)
                                                        std::cout << "print LineValue ";

                                                sprintf(a4, "%s", tst->attrib);
                                                rec.LineNum = true;
                                                rec.LineValue = true;
                                                rec.Console = true;

                                        }

                                }


                                tst = Lookup(a5);
                                if (tst != NULL )
                                {
                                        if (tst->type != FileVar)
                                        {
                                std::cout <<  "invalid type ...  " <<  a5 << " FileVar
Expected" << std::endl;
```

```
                                        err = true;
                                }
                                else
                                        sprintf(a5, "%s", tst->attrib);

                        }

                        if (a5 != NULL && !err)
                        {
                                std::cout << "Output to File " << a5 << std::endl;

                                sprintf(rec.FileStr, "%s", a5);
                                rec.File = true;

                        }

                        tst = Lookup(a6);
                        if (tst != NULL )
                        {
                                if (tst->type != StrVar)
                                {
                        std::cout <<  "LineValue keyword expected" << std::endl;
                                        err = true;
                                }
                                else
                                {
                                        if(strcmp(tst->attrib, "LineValue") == 0)
                                                std::cout << "Output to  LineValue "
<< std::endl;

                                        sprintf(a6, "%s", tst->attrib);

                                }

                        }


                        if (!err)
                        {
                                std::cout << "Emit Code for ReplaceStr("
                                        << a1 << ","
                                        << a2 << ","
                                        << a3 << ","
                                        << "case = " << cas << ","
                                        << "subDir = " << subdir << ")"
                                        << std::endl;
```

```
                                        ReplaceStr(a1, a2, a3, cas, subdir, &rec);

                                }

                                free (a1);
                                free(a2);
                                free (a3);
                                free(a4);
                                free(a5);
                                free(a6);
                        }
        ;


stmt7 { char *a1, *a2, *a3; bool cas, subdir;}
:  #("Replace" a1=strORid a2=strORid  a3= strORid cas = caseOnOff
                        subdir = subDirOnOff)
                        {
                                bool err = false;
                                PrnRec rec;

                                initPrnRec(&rec);

                                struct _Symbol *tst = NULL;
                                tst = Lookup(a1);
                                if (tst != NULL)
                                {
                std::cout << "String re-decleared ...       " << a1 << std::endl;
                                        err=true;
                                }
                                if (tst->type != StrVar)
                                {
                std::cout << "invalid type ...       " <<  a1 <<  "StrVar expected" << std::endl;
                                        err = true;
                                }

                                tst = Lookup(a2);
                                if (tst != NULL )
                                {
                std::cout << "String re-decleared ...       " << a2 << std::endl;
                                        err=true;
                                }
                                if (tst->type != StrVar)
                                {
                std::cout << "invalid type ...       " << a2<<" StrVar Expected" << std::endl;
```

```
                                        err = true;
                                }

                                tst = Lookup(a3);
                                if (tst != NULL )
                                {
                std::cout << "File re-decleared ...          " << a3 << std::endl;
                                        err=true;
                                }
                                if (tst->type != FileVar)
                                {
                std::cout << "invalid type ...       " <<  a3 << "File type Expected" <<
std::endl;
                                        err = true;
                                }


                                if (!err)
                                {
                                        std::cout << "Emit Code for ReplaceStr("
                                                << a1 << ","
                                                << a2 << ","
                                                << a3 << ","
                                                << "case = " << cas << ","
                                                << "subDir = " << subdir << ")"
                                                << std::endl;
                                }

                                free (a1);
                                free(a2);
                                free(a3);
                        }
        ;

strORid returns [char *s]:  { s = (char *)malloc(80); }
                                                        s1:STRING
                                                        {
                                                                sprintf(s, "%s", s1-
>getText().c_str());
                                                        }
                                                        | { s = (char *)malloc(80); }
                                                          i1:ID
                                                          {
                                                                        sprintf(s,"%s",i1-
>getText().c_str());
```

```
                                                                    struct _Symbol *sym
= Lookup(s);
                                                                    if ( sym == NULL)
                                                                            std::cout <<
"identifier " << s << " not defined" << std::endl;
                                                                    // else
                                                                    //        sprintf(s,
"%s", sym->attrib);
                                                            }
                        ;

caseOnOff returns [bool cas] : "CaseOn" { cas = true; }
                                                                    | "CaseOff" { cas =
false; }
                                                                    ;

subDirOnOff returns [bool subdir] : "SubDirectoryOn" { subdir = true; }
                                                                            |
"SubDirectoryOff" { subdir = false; }
                                                                            ;

// END OF TREEWALKER FOR STATIC SEMATIC CHECKING
```

symbol.h

The header file for the SSL symbol table.

```
/*                                      */
/*   Header file for SSL Symbol Table       */
/*     author: Meera Ganesan              */
/*                                      */

#ifndef INC_SYMBOL_H
#define INC_SYMBOL_H


/*#define  DEBUG */

/* Symbol table Size */
#define  SymbolTableSize          100

/* function return status */
#define  STATUS_SUCCESS            0
#define  STATUS_FAILURE           -1

/* Symbol type definitions */
#define  Constant                  0
#define  Literals                  1
#define  FileVar                   2
#define  DirVar                    3
#define  StrVar                    4

/* Structure definitions */
struct  _Symbol {
        char   *name;                   /* symbol name  */
                                        int    type;                /* Symbol type variable, file name
                                        char   *attrib;              /* attrib if any that needs to be sa
                                                                struct _Symbol *next;
} ;


extern int Insert(char *tokname, int toktype, char *tokattrib);
extern struct _Symbol *Lookup(char *s);


#endif
```

symbol.c

The C code for the SSL symbol table.

```
/*                          */
/*   SSL Symbol Table Code      */
/*     author: Meera Ganesan     */
/*                          */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "symbol.h"

/* Linked List definition */
static struct _Symbol *symboltable[SymbolTableSize];

static int ListHeader;                  /* LinkedList Header */

/*
void main()
{
  struct _Symbol *st;
  int status = 0;

#ifdef  DEBUG
  // some e.g. to insert and do look up in symbol table
  char *token1="TOKEN1";
  int  toktype1 = Constant;
  char *tokattrib1 = "code1";
  char *token2="TOKEN2";
  int  toktype2 = Variable;
  char *tokattrib2 = "code2";
#endif                  // DEBUG

   * make the Symbol table empty *
   ListHeader=0;

#ifdef  DEBUG
   status = Insert(token1, toktype1, tokattrib1);
   status = Insert(token2, toktype2, tokattrib2);
```

```
#endif                  // DEBUG
   return ;
}

*/
/*****************************************************
Insert()
 **Input  Synbol
         Symbol type
                  Symbol attribute

 ** Output Insert status
*****************************************************/
int Insert(char *tokname, int toktype, char *tokattrib)
{
    struct _Symbol *tst = NULL; /* *Lookup(); */

          /* check for Overflow status */
    if (ListHeader >= SymbolTableSize) {
#ifdef DEBUG
            printf ("\n Symbol Table Link List overflow ....");
#endif
            return(STATUS_FAILURE);
          }

          /* Check to see if the symbol is already present */
          if ((tst = Lookup(tokname)) == NULL ) {
       tst = (struct _Symbol *)malloc(sizeof(*tst));
            tst->type = toktype;
            tst->name = malloc(strlen(tokname+1));
            strcpy(tst->name, tokname);
            tst->attrib = malloc(strlen(tokattrib+1));
            strcpy(tst->attrib, tokattrib);

       tst->next = symboltable[ListHeader];
            symboltable[ListHeader++]=tst;

#ifdef DEBUG
            printf("\n Adding %s to symbol table", tst->name);
       printf("\n  %d  %s\n", ListHeader, symboltable[ListHeader-1]->name);
#endif
       return(STATUS_SUCCESS);
    }
          else {
#ifdef DEBUG
            printf("\n Symbol Already exists ....");
```

```
#endif
   return(STATUS_FAILURE);
        }
}
/*****************************************************
Lookup()
 **Input  Symbol

 ** Output
       symbol
                symbol type
                symbol attribute
*****************************************************/
struct _Symbol *Lookup(char *s)
{
    struct _Symbol *nSym = NULL;
         int  i = 0;

#ifdef DEBUG
    printf("I am looking for %s", s);
#endif
         if ( s == NULL)
                return(NULL);

    for (i=0; i< ListHeader; i++) {
            nSym = symboltable[i];
                if (strcmp(s, nSym->name) == 0) {
#ifdef DEBUG
                   printf("String exits.... ");
#endif
                        return(nSym);
                }
         }
         return(NULL);
}
```

SSLPkg.h

The header file for the C++ Libraries used in code generation.

```
//
// Header file for SSL Compiler C++ Code Generation
//   author: Dennis Kim
//

#ifndef FINDSTR_H
#define FINDSTR_H

//#include <windows.h>

struct PrnRec{
        bool    File;               // for Output to a file
        char    FileStr[256];   // name of the output file
        bool    Console;        // for Print to the screen
        bool    DirName;
        bool    FileName;
        bool    LineNum;
        bool    LineValue;
};

extern char FILENAME[256];                  // global file name

void initPrnRec(PrnRec *rec);

void FindStr(char *, char *, bool, bool, PrnRec * pRec);
void ReplaceStr(char * str1, char * str2, char * str3, bool cas, bool subdir, PrnRec *
pRec);

void emitCodeForOutput(char *str1, char *str2);
void emitCodeForPrint(char *str1);

#endif
```

SSLPkg.cpp

The C++ Libraries used in code generation.

```
//
// SSL Code Generation Libraries
//  author: Dennis Kim
//

#include <fstream.h>
#include <string.h>

#include "SSLPkg.h"

static int count = 0;

void initPrnRec(PrnRec *rec)
{
        rec->Console = false;
        rec->File = false;
        rec->LineNum = false;
        rec->LineValue = false;
        rec->FileName = true;
        rec->DirName = true;
        strcpy(rec->FileStr, "");

}

void FindStr(char * str1, char * str2, bool cas, bool subdir, PrnRec * pRec)
{
 count++;
 ofstream TargetFile(FILENAME, ios::app);

 TargetFile << "\nPrnRec tempPrnRec" << count << " = {";
 pRec->File ? TargetFile << "true, " : TargetFile << "false, ";
 TargetFile << "\"" << pRec->FileStr << "\", ";
 pRec->Console ? TargetFile << "true, " : TargetFile << "false, ";
 pRec->DirName ? TargetFile << "true, " : TargetFile << "false, ";
 pRec->FileName ? TargetFile << "true, " : TargetFile << "false, ";
 pRec->LineNum ? TargetFile << "true, " : TargetFile << "false, ";
 pRec->LineValue ? TargetFile << "true}" : TargetFile << "false};";
 TargetFile << endl;

 TargetFile << "\nFindStr(\"" << str1 << "\", \"" << str2 << "\", ";
```

```
  cas ? TargetFile << "true, " : TargetFile << "false, ";
  subdir ? TargetFile << "true, " : TargetFile << "false, ";
  TargetFile << "&tempPrnRec" << count << ");" << endl;

  TargetFile.close();
}


void ReplaceStr(char * str1, char * str2, char * str3, bool cas, bool subdir, PrnRec *
pRec)
{
  count++;
  ofstream TargetFile(FILENAME, ios::app);

  TargetFile << "\nPrnRec tempPrnRec" << count << " = {";
  pRec->File ? TargetFile << "true, " : TargetFile << "false, ";
  TargetFile << "\"" << pRec->FileStr << "\", ";
  pRec->Console ? TargetFile << "true, " : TargetFile << "false, ";
  pRec->DirName ? TargetFile << "true, " : TargetFile << "false, ";
  pRec->FileName ? TargetFile << "true, " : TargetFile << "false, ";
  pRec->LineNum ? TargetFile << "true, " : TargetFile << "false, ";
  pRec->LineValue ? TargetFile << "true}" : TargetFile << "false};";
  TargetFile << endl;

  TargetFile << "\nReplaceStr(\"" << str1 << "\", \"" << str2 << "\", " << str3 << "\", ";
  cas ? TargetFile << "true, " : TargetFile << "false, ";
  subdir ? TargetFile << "true, " : TargetFile << "false, ";
  TargetFile << "&tempPrnRec" << count << ");" << endl;

  TargetFile.close();
}

void emitCodeForOutput(char *str1, char *str2)
{

        static bool output_file_open = false;

        ofstream TargetFile(FILENAME, ios::app);

        if (output_file_open == false)
        {
                TargetFile << "\t ofstream outputFile;" << endl;
                output_file_open = true;
        }

        TargetFile << "\t outputFile.open(\"" << str1 << "\"," << "ios:app);" << endl;
```

```
        TargetFile << "\t outputFile << \"" << str2 << "\"" << "<< endl;" << endl;
        TargetFile << "\t outputFile.close();" << endl;

        TargetFile.close();

}

void emitCodeForPrint(char *str1)
{

        ofstream TargetFile(FILENAME, ios::app);

        TargetFile << "\t std::cout << \"" << str1 << "\"" << "<< endl;" << endl;

        TargetFile.close();

}
```

SSLLib.h

The header file for SSL compilation which pulls in the C++ libraries needed.

```
//
// SSL Code Compilation Libraries
//  author: Dennis Kim
//

#ifndef SSLLIB_H
#define SSLLIB_H


struct PrnRec{
        bool    File;           // for Output to a file
        char *FileStr;  // name of the output file
        bool    Console;        // for Print to the screen
        bool    DirName;
        bool    FileName;
        bool    LineNum;
        bool    LineValue;
};


void FindStr(char * str1, char * str2, bool cas, bool subdir, PrnRec * pRec);

void ReplaceStr(char * sourceStr, char * locStr, char * destStr, bool cas, bool subdir,
PrnRec * pRec);



#endif
```

SSLLib.cpp

The C++ Libraries used for SSL compilation which pulls in the C++ libraries needed.

```
//
// SSL Libraries
//  Must be linked with SSL program for compile
//  author: Dennis Kim
//
//

#include <fstream.h>
#include <string.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>
#include <direct.h>
#include "SSLLib.h"

/*

void FindStr(char * str1, char * str2, bool cas, bool subdir, PrnRec * pRec)
{
 char CurrentLine[256];
 char CAPstr[256];
 char CAPline[256];
 _finddata_t my_file;
 char Pathname[_MAX_PATH];
 char Drive[_MAX_DRIVE];
 char Dir[_MAX_DIR];
 char filename[_MAX_FNAME];
 char Ext[_MAX_EXT];
 long hFile;

 _splitpath(str1, Drive, Dir, filename, Ext);
 hFile = _findfirst(str1, &my_file);

 while(errno == 0)
 {
  if (strcmp(my_file.name, ".") != 0  &&
      strcmp(my_file.name, "..") != 0 &&
      !(my_file.attrib & _A_SUBDIR)   &&
```

```
    stricmp(Pathname, pRec->FileStr) != 0)
  {
   sprintf(Pathname, "%s%s%s", Drive, Dir, my_file.name);
   ifstream CurrentFile(Pathname, ios::in | ios::nocreate);

   if (CurrentFile.is_open())
   {
    int count = 0;
    while(!CurrentFile.eof())
    {
     count++;
     CurrentFile.getline(CurrentLine, 256);

     if (!cas)
     {
      strcpy(CAPstr, str2);
      strcpy(CAPline, CurrentLine);
      _strupr(CAPstr);
      _strupr(CAPline);
     }

     if ((cas && (strstr(CurrentLine, str2) != 0)) ||
      (!cas && (strstr(CAPline, CAPstr) != 0)))
     {
      if (pRec->File)
      {
       ofstream OutFile(pRec->FileStr, ios::out | ios::app);
       if (pRec->DirName)
        OutFile << Drive << Dir;
       if (pRec->FileName)
        OutFile << my_file.name;
       if (pRec->LineNum)
        OutFile << "(" << count << "):";
       if (pRec->LineValue)
        OutFile << "    " << CurrentLine;
       OutFile << endl;
       OutFile.close();
      }
      if (pRec->Console)
      {
       if (pRec->DirName)
        cout << Drive << Dir;
       if (pRec->FileName)
        cout << my_file.name;
       if (pRec->LineNum)
        cout << "(" << count << "):";
```

```
            if (pRec->LineValue)
              cout << "    " << CurrentLine;
            cout << endl;
          }
        }
      }
    }

    CurrentFile.close();
   }
   _findnext(hFile, &my_file);
   sprintf(Pathname, "%s%s%s", Drive, Dir, my_file.name);
  }

  _findclose(hFile);
}



void ReplaceStr(char * sourceStr, char * locStr, char * destStr, bool cas, bool subdir,
PrnRec * pRec)
{
  char CurrentLine[256];
  char CAPstr[256];
  char CAPline[256];
  _finddata_t my_file;
  char Pathname[_MAX_PATH];
  char Drive[_MAX_DRIVE];
  char Dir[_MAX_DIR];
  char filename[_MAX_FNAME];
  char Ext[_MAX_EXT];
  long hFile;
  char * tempStr;
  char tempArr[256];
  char tempArr2[256];
  int skip;
  bool found;

  _splitpath(locStr, Drive, Dir, filename, Ext);
  hFile = _findfirst(locStr, &my_file);

  while(errno == 0)
  {
   if (strcmp(my_file.name, ".") != 0  &&
       strcmp(my_file.name, "..") != 0 &&
       !(my_file.attrib & _A_SUBDIR)   &&
```

```
    stricmp(Pathname, pRec->FileStr) != 0)
  {
   sprintf(Pathname, "%s%s%s", Drive, Dir, my_file.name);
   ifstream CurrentFile(Pathname, ios::in | ios::out);
   char tempFileName[256];
   sprintf(tempFileName, "%s%s%s", Drive, Dir, "yo_yo.zzz");
   ofstream TempFile(tempFileName, ios::out);

   if (CurrentFile.is_open())
   {
    int count = 0;
    while(!CurrentFile.eof())
    {
     count++;
     CurrentFile.getline(CurrentLine, 256);

     strcpy(tempArr2, CurrentLine);

     found = false;
     do
     {
      if (!cas)
      {
       strcpy(CAPstr, sourceStr);
       strcpy(CAPline, tempArr2);
       _strupr(CAPstr);
       _strupr(CAPline);
      }

      if (cas)
      {
       tempStr = strstr(tempArr2, sourceStr);
       if (tempStr != 0)
         skip = tempStr - tempArr2;
      }
      else
      {
       tempStr = strstr(CAPline, CAPstr);
       if (tempStr != 0)
         skip = tempStr - CAPline;
      }

      if (tempStr != 0)
      {
       found = true;
       strncpy(tempArr, tempArr2, skip);
```

```
          tempArr[skip] = '\0';
          strcat(tempArr, destStr);
          strcat(tempArr, tempArr2 + skip + strlen(sourceStr));
          strcpy(tempArr2, tempArr);
        }
      } while (tempStr != 0);


      if (found)
      {
       TempFile << tempArr << endl;
       if (pRec->File)
       {
        ofstream OutFile(pRec->FileStr, ios::out | ios::app);
        if (pRec->DirName)
          OutFile << Drive << Dir;
        if (pRec->FileName)
          OutFile << my_file.name;
        if (pRec->LineNum)
          OutFile << "(" << count << "):";
        if (pRec->LineValue)
          OutFile << "    " << tempArr;
        OutFile << endl;
        OutFile.close();
       }
       if (pRec->Console)
       {
        if (pRec->DirName)
          cout << Drive << Dir;
        if (pRec->FileName)
          cout << my_file.name;
        if (pRec->LineNum)
          cout << "(" << count << "):";
        if (pRec->LineValue)
          cout << "    " << tempArr;
        cout << endl;
       }
      }
      else
       TempFile << CurrentLine << endl;
    }
  }

  CurrentFile.close();
  TempFile.close();
  remove(Pathname);
```

```
      rename(tempFileName, Pathname);
    }
    _findnext(hFile, &my_file);
    sprintf(Pathname, "%s%s%s", Drive, Dir, my_file.name);
  }

  _findclose(hFile);
}
*/
```

Makefile

The Unix Makefile for SSL.

```
#Makefile for SSL C++

#Author: Sandy MacDonald
#
INCL = -I/home/slm154/plt/include

INCL1 = -L/home/slm154/plt/lib

LIST = symbol.o SSLParser.o SSLLexer.o SSLWalker.o SSLPkg.o sslc.o

#

#Link sslc (Main)

#

sslc: $(LIST)

                        c++ -o sslc $(LIST) $(INCL1) -lantlr


#
#Compiles

#

sslc.o: sslc.cpp

                            c++ -c sslc.cpp $(INCL)

#
symbol.o: symbol.c

                                cc -c symbol.c

#

SSLParser.o: SSLParser.cpp
```

```
	c++ -c SSLParser.cpp $(INCL)

#

SSLLexer.o: SSLLexer.cpp

	c++ -c SSLLexer.cpp $(INCL)

#

SSLWalker.o: SSLWalker.cpp

	c++ -c SSLWalker.cpp $(INCL)

#

SSLPkg.o: SSLPkg.cpp

	c++ -c SSLPkg.cpp
```

README.txt

The readme file for the SSL zip file.

SSL Project Team — Programming Languages & Translators

CSW4115(CVN) Spring 2003 — Professor: Stephen Edwards

SSL Team:
Meera Ganesan meera.ganesan@intel.com
Dennis Kim dkim@harris.com
Sandy MacDonald SandraMacDonald@att.com
Satheesha Rangegowda Satheesha_rangegowda_923@agilent.com

SSL String Searching Language Compiler

Function: Program sslc.exe is a compiler for the SSL language.
It takes an SSL program (with file extension .ssl) and produces a .cpp.
This output files can be compiled using g++ and linking with SSLLib.

SSL.Zip file includes: sslc.exe, sslc.cpp, SSLLexer.cpp, SSLLexer.hpp,
SSLParser.cpp, SSLParser.hpp, SSLWalker.cpp, SSLWalker.hpp,
SSLLexerTokenTypes.hpp, SSLLexerTokenTypes.cpp, symbol.c, symbol.h,
SSLPkg.c, SSLPkg.h, SSLLib.c, SLLib.h, this README.txt and a Unix Makefile.

SSL Compilation Instructions:
Type "sslc xxxx.ssl" where xxxx is the name of the SSL program to compile.
Compile the resulting xxxx.cpp program using g++ and link with SSLLib.

****

For Future Use*
SSLC Build Instructions: Unzip and run make in library /xxxx such that
the ANTLR code libraries (/xxxx/antlr-2.7.1 and /xxxx/lib/cpp) reside
under the current directory.
Requires ANTLR-2.7.1. See www.antlr.org for download.

Requires PATH set to $PATH:/xxxx/antlr-2.7.1:/xxxx/lib/cpp
Requires update to Makefile to indicate path where antlr is installed.
Note: Use of downlevel version of ANTLR code since most current version (ANTLR-2.7.2) does not support C++ without errors.
Note: See 1st Caveat and Defect - This release does not run on Unix.

****

Known Caveats and Defects with V1.0:
-This release is limited to DOS and does not run on Unix.
-Concatenation is not supported with this release.
-Readline is not supported in this release.
-If both output and print appear in the same scope, they must have the same format.
-The SubDirectoryOn parameter is always set to false.
-Makefile does not automatically pick up the antlr directory path or take as an input parameter.