**Assignment 4: The Parser**

#V

FOR: Svillen Ranev

Due: August 12th , 2011

CST8152 – Compilers

Lab Section 11

Email 1: coll0300@algonquinlive.com

Email 2: curtis\_collins\_1@hotmail.com

**Contents:**

Grammar

parser.h

parser.c

Test Results

Test Plan

Bonus Explanation

/\* File Name: parser.h

\* Version: 1.0

\* Author: William Collins (040652633)

\* Course: CST8152 - Compilers

\* Assignment: 4

\* Date: August 12, 2011

\* Professor: Svillen Ranev

\* Purpose: Declarations and prototypes for the parser

\*/

#ifndef PARSER\_H

#define PARSER\_H

/\*CONSTANT DEFINITIONS\*/

#define NO\_ATTR -1

#define ELSE 0

#define IF 1

#define INPUT 2

#define OUTPUT 3

#define PLATYPUS 4

#define REPEAT 5

#define THEN 6

#define USING 7

/\*GLOBAL VARIABLE DEFINITIONS\*/

static Token lookahead\_token; /\*The next token to be parsed\*/

static Buffer \*sc\_buf; /\*buffer that holds the source\*/

extern STD sym\_table; /\*symbol table\*/

extern Buffer \*str\_LTBL; /\*Holds string literals found in the source\*/

extern int line; /\*Line numbers\*/

extern char \*kw\_table[]; /\*The keyword table declared in table.h\*/

int synerrno; /\*Number of errors found\*/

/\*FUNCTION PROTOTYPES\*/

void parser(Buffer \*);

void match(int, int);

void syn\_eh(int);

void syn\_printe(void);

void gen\_incode(const char \*);

Token malpar\_next\_token(Buffer \*);

/\*GRAMMAR FUNCTIONS\*/

void program(void);

void opt\_statements(void);

void statements(void);

void statements1(void);

void statement(void);

void assignment\_statement(void);

void assignment\_expression(void);

void selection\_statement(void);

void iteration\_statement(void);

void input\_statement(void);

void variable\_list(void);

void variable\_list1(void);

void variable\_identifier(void);

void output\_statement(void);

void output\_statement1(void);

void opt\_variable\_list(void);

void arithmetic\_expression(void);

void unary\_arithmetic\_expression(void);

void additive\_arithmetic\_expression(void);

void additive\_arithmetic\_expression1(void);

void multiplicative\_arithmetic\_expression(void);

void multiplicative\_arithmetic\_expression1(void);

void primary\_arithmetic\_expression(void);

void string\_expression(void);

void string\_expression1(void);

void primary\_string\_expression(void);

void conditional\_expression(void);

void logical\_or\_expression(void);

void logical\_or\_expression1(void);

void logical\_and\_expression(void);

void logical\_and\_expression1(void);

void relational\_expression(void);

void primary\_a\_relational\_expression(void);

void primary\_a\_relational\_expression1(void);

void primary\_s\_relational\_expression(void);

void primary\_s\_relational\_expression1(void);

#endif

/\* File Name: parser.c

\* Version: 1.0

\* Author: William Collins (040652633)

\* Course: CST8152 - Compilers

\* Assignment: 4

\* Date: August 12, 2011

\* Professor: Svillen Ranev

\* Purpose: Parses a token represented source file

\*/

#include <stdio.h>

#include <stdlib.h>

#include "buffer.h"

#include "token.h"

#include "stable.h"

#include "parser.h"

/\*

\* Purpose: scans and parses the source file contained in a buffer

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: in\_buf - The buffer containing the source file

\*/

void parser(Buffer \*in\_buf){

sc\_buf = in\_buf;

lookahead\_token = malpar\_next\_token(sc\_buf);

program();

match(SEOF\_T,NO\_ATTR);

gen\_incode("PLATY: Source file parsed");

}

/\*

\* Purpose: Matches the lookahead token with the expected token

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: pr\_token\_code - The expected token

pr\_token\_attribute - the expected attribute of the token

\*/

void match(int pr\_token\_code,int pr\_token\_attribute) {

int code = lookahead\_token.code; /\*makes a shorter if statement\*/

/\*We know for sure that there is an error\*/

if (code != pr\_token\_code) {

syn\_eh(pr\_token\_code);

return;

}

/\*Test further for attribute equality.\*/

if (code == KW\_T && pr\_token\_attribute != lookahead\_token.attribute.int\_value

|| code == REL\_OP\_T && pr\_token\_attribute != lookahead\_token.attribute.rel\_op

|| code == LOG\_OP\_T && pr\_token\_attribute != lookahead\_token.attribute.log\_op

|| code == ART\_OP\_T && pr\_token\_attribute != lookahead\_token.attribute.arr\_op )

{

syn\_eh(pr\_token\_code);

return;

}

/\*Handle End of File before the scanner generates an error\*/

if (code == SEOF\_T){

return;

}

lookahead\_token = malpar\_next\_token(sc\_buf);

/\*handle errors caught in the scanner\*/

if (lookahead\_token.code == ERR\_T){

syn\_printe();

lookahead\_token = malpar\_next\_token(sc\_buf);

synerrno++;

return;

}

}

/\*

\* Purpose: Handles a panic mode error recovery

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: sync\_token\_code - The code of our safety net token

\*/

void syn\_eh(int sync\_token\_code){

/\*Print the error and increment the error count\*/

syn\_printe();

synerrno++;

/\*If we are looking for end of file, let's just leave early\*/

if (sync\_token\_code == SEOF\_T){

return;

}

/\*sync up the parser by finding the next occurence of sync token\*/

while (lookahead\_token.code != sync\_token\_code) {

lookahead\_token = malpar\_next\_token(sc\_buf);

if (lookahead\_token.code == SEOF\_T){

exit(synerrno);

}

}

/\*let's advance once more now that we know we are safe\*/

lookahead\_token = malpar\_next\_token(sc\_buf);

}

/\*

\* Purpose: Generates intermediate code. (This implementation only prints)

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: code - A string of code that is added to the generated code

\*/

void gen\_incode(const char \*code){

printf("%s\n", code);

}

/\*

\* Purpose: Prints out an error found by the parser

\* Author: Svillen Ranev

\* History/Versions: Unknown

\*/

void syn\_printe() {

Token t = lookahead\_token;

printf("PLATY: Syntax error: Line:%3d\n",line);

printf("\*\*\*\*\* Token code:%3d Attribute: ", t.code);

switch(t.code){

case ERR\_T: /\* ERR\_T 0 Error token \*/

printf("%s\n",t.attribute.err\_lex);

break;

case SEOF\_T: /\*SEOF\_T 1 Source end-of-file token \*/

printf("NA\n" );

break;

case AVID\_T: /\* AVID\_T 2 Arithmetic Variable identifier token \*/

case SVID\_T :/\* SVID\_T 3 String Variable identifier token \*/

printf("%s\n",sym\_table.pstvr[t.attribute.get\_int].plex);

break;

case FPL\_T: /\* FPL\_T 4 Floating point literal token \*/

printf("%5.1f\n",t.attribute.flt\_value);

break;

case INL\_T: /\* INL\_T 5 Integer literal token \*/

printf("%d\n",t.attribute.get\_int);

break;

case STR\_T:/\* STR\_T 6 String literal token \*/

printf("%s\n",(str\_LTBL->ca\_head + t.attribute.get\_int));

break;

case SCC\_OP\_T: /\* 7 String concatenation operator token \*/

printf("NA\n" );

break;

case ASS\_OP\_T:/\* ASS\_OP\_T 8 Assignment operator token \*/

printf("NA\n" );

break;

case ART\_OP\_T:/\* ART\_OP\_T 9 Arithmetic operator token \*/

printf("%d\n",t.attribute.get\_int);

break;

case REL\_OP\_T: /\*REL\_OP\_T 10 Relational operator token \*/

printf("%d\n",t.attribute.get\_int);

break;

case LOG\_OP\_T:/\*LOG\_OP\_T 11 Logical operator token \*/

printf("%d\n",t.attribute.get\_int);

break;

case LPR\_T: /\*LPR\_T 12 Left parenthesis token \*/

printf("NA\n" );

break;

case RPR\_T: /\*RPR\_T 13 Right parenthesis token \*/

printf("NA\n" );

break;

case LBR\_T: /\* 14 Left brace token \*/

printf("NA\n" );

break;

case RBR\_T: /\* 15 Right brace token \*/

printf("NA\n" );

break;

case KW\_T: /\* 16 Keyword token \*/

printf("%s\n",kw\_table [t.attribute.get\_int]);

break;

case COM\_T: /\* 17 Comma token \*/

printf("NA\n");

break;

case EOS\_T: /\* 18 End of statement \*(semi - colon) \*/

printf("NA\n" );

break;

default:

printf("PLATY: Scanner error: invalid token code: %d\n", t.code);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A FUNCTION FOR EVERY DEFINED PRODUCTION IN THE GRAMMAR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* PRODUCTION: <program> -> PLATYPUS {<opt\_statements>}

\*FIRSTS: PLATYPUS

\*/

void program() {

match(KW\_T,PLATYPUS);

match(LBR\_T,NO\_ATTR);

opt\_statements();

match(RBR\_T,NO\_ATTR);

gen\_incode("PLATY: Program parsed");

}

/\*

\* PRODUCTION: <opt\_statements> -> <statements> | empty

\* FIRSTS: AVID\_T, SVID\_T, IF, USING, INPUT, OUTPUT, empty

\*/

void opt\_statements(){

int c = lookahead\_token.code;

int a = lookahead\_token.attribute.int\_value;

/\*The FIRSTS for opt\_statements\*/

if (c == AVID\_T || c == SVID\_T || c == KW\_T && (a == IF || a == USING || a == INPUT || a == OUTPUT)){

statements();

}

/\*empty\*/

else {

gen\_incode("PLATY: Opt\_statements parsed");

}

}

/\*

\* PRODUCTION: <statements> -> <statement><statements'>

\* FIRSTS: AVID\_T, SVID\_T, IF, USING, INPUT, OUTPUT

\*/

void statements(void){

statement();

statements1();

}

/\*

\* PRODUCTION: <statements1> -> <statement><statements1> | empty

\* FIRSTS: AVID\_T, SVID\_T, IF, USING, INPUT, OUTPUT

\*/

void statements1(void){

int c = lookahead\_token.code;

int a = lookahead\_token.attribute.int\_value;

if (c == AVID\_T || c == SVID\_T || c == KW\_T && (a == IF || a == USING || a == INPUT || a == OUTPUT)){

statement();

statements1();

}

/\*empty\*/

}

/\*

\* PRODUCTION: <statement> -> <assignment statement> | <selection statement> |

<iteration statement> | <input statement> | <output statement>

\* FIRSTS: AVID\_T, SVID\_T, IF, USING, INPUT, OUTPUT

\*/

void statement(void){

/\*Assignment statement\*/

if (lookahead\_token.code == AVID\_T || lookahead\_token.code == SVID\_T){

assignment\_statement();

}

/\*selection statement\*/

else if (lookahead\_token.attribute.get\_int == IF) {

selection\_statement();

}

/\*iteration statement\*/

else if (lookahead\_token.attribute.get\_int == USING){

iteration\_statement();

}

/\*input statement\*/

else if (lookahead\_token.attribute.get\_int == INPUT){

input\_statement();

}

/\*output statement\*/

else if (lookahead\_token.attribute.get\_int == OUTPUT){

output\_statement();

}

/\*Error\*/

else {

syn\_printe();

return;

}

}

/\*

\* PRODUCTION: <assignment statement> -> <assignment expression>

\* FIRSTS: AVID\_T, SVID\_T

\*/

void assignment\_statement(){

assignment\_expression();

match(EOS\_T, NO\_ATTR);

gen\_incode("PLATY: Assignment statement parsed");

}

/\*

\* PRODUCTION: <assignment expression> -> AVID = <arithmetic expression>

| SVID = <arithmetic expression>

\* FIRSTS: AVID\_T, SVID\_T

\*/

void assignment\_expression(){

/\*AVID\*/

if (lookahead\_token.code == AVID\_T) {

match(AVID\_T, NO\_ATTR);

match(ASS\_OP\_T, NO\_ATTR);

arithmetic\_expression();

gen\_incode("PLATY: Assignment expression (arithmetic) parsed");

}

/\*SVID\*/

else if (lookahead\_token.code == SVID\_T){

match(SVID\_T, NO\_ATTR);

match(ASS\_OP\_T, NO\_ATTR);

string\_expression();

gen\_incode("PLATY: Assignment expression (string) parsed");

}

/\*Error\*/

else {

syn\_printe();

}

}

/\*

\* PRODUCTION: <selection statement> -> IF (<conditional expression>) THEN <opt\_statements>

ELSE {<opt\_statements>};

\* FIRSTS: IF

\*/

void selection\_statement(){

match(KW\_T, IF);

match(LPR\_T, NO\_ATTR);

conditional\_expression();

match(RPR\_T, NO\_ATTR);

match(KW\_T, THEN);

opt\_statements();

match(KW\_T, ELSE);

match(LBR\_T, NO\_ATTR);

opt\_statements();

match(RBR\_T, NO\_ATTR);

match(EOS\_T, NO\_ATTR);

gen\_incode("PLATY: IF statement parsed");

}

/\*

\* PRODUCTION: <iteration statement> -> USING (<assignment expression>,<conditional expression>,<assignment expression>)

REPEAT { <opt\_statements> };

\* FIRSTS: USING

\*/

void iteration\_statement(){

match(KW\_T, USING);

match(LPR\_T, NO\_ATTR);

assignment\_expression();

match(COM\_T, NO\_ATTR);

conditional\_expression();

match(COM\_T, NO\_ATTR);

assignment\_expression();

match(RPR\_T, NO\_ATTR);

match(KW\_T, REPEAT);

match(LBR\_T, NO\_ATTR);

opt\_statements();

match(RBR\_T, NO\_ATTR);

match(EOS\_T, NO\_ATTR);

gen\_incode("PLATY: USING statement parsed");

}

/\*

\* PRODUCTION: <input statement> -> INPUT(<variable list>);

\* FIRSTS: INPUT

\*/

void input\_statement(){

match(KW\_T, INPUT);

match(LPR\_T, NO\_ATTR);

variable\_list();

match(RPR\_T, NO\_ATTR);

match(EOS\_T, NO\_ATTR);

gen\_incode("PLATY: INPUT statement parsed");

}

/\*

\* PRODUCTION: <variable list> -> <variable identifier><variable list1>

\* FIRSTS: AVID\_T, SVID\_T

\*/

void variable\_list(){

variable\_identifier();

variable\_list1();

gen\_incode("PLATY: Variable list parsed");

}

/\*

\* PRODUCTION: <variable list1> -> ,<variable identifier><variable list1> | empty

\* FIRSTS: ,, empty

\*/

void variable\_list1(){

/\*empty\*/

if (lookahead\_token.code != COM\_T) {

return;

}

match(COM\_T, NO\_ATTR);

variable\_identifier();

variable\_list1();

}

/\*

\* PRODUCTION: <variable identifier> -> AVID\_T | SVID\_T

\* FIRSTS: AVID\_T, SVID\_T

\*/

void variable\_identifier(){

if (lookahead\_token.code == AVID\_T){

match(AVID\_T, NO\_ATTR);

}

else if (lookahead\_token.code == SVID\_T){

match(SVID\_T, NO\_ATTR);

}

else {

syn\_printe();

}

}

/\*

\* PRODUCTION: <output statement> -> OUTPUT(<output statement1>);

\* FIRSTS: OUTPUT

\*/

void output\_statement(){

match(KW\_T, OUTPUT);

match(LPR\_T, NO\_ATTR);

output\_statement1();

match(RPR\_T, NO\_ATTR);

match(EOS\_T, NO\_ATTR);

gen\_incode("PLATY: OUTPUT statement parsed");

}

/\*

\* PRODUCTION: <output statement1> -> <opt\_variable list> | STR\_T

\* FIRSTS: AVID\_T, SVID\_T, STR\_T, empty

\*/

void output\_statement1(){

/\*String literal\*/

if (lookahead\_token.code == STR\_T){

match(STR\_T, NO\_ATTR);

gen\_incode("PLATY: Output list (string literal) parsed");

}

/\*Optional Variable list\*/

else {

opt\_variable\_list();

}

}

/\*

\* PRODUCTION: <opt\_variable list> -> <variable list> | empty

\* FIRSTS: AVID\_T, SVID\_T, empty

\*/

void opt\_variable\_list(){

/\*variable list\*/

if (lookahead\_token.code == AVID\_T || lookahead\_token.code == SVID\_T){

variable\_list();

}

/\*empty\*/

else {

gen\_incode("PLATY: Output list (empty) parsed");

}

}

/\*

\* PRODUCTION: <arithmetic expression> -> <unary arithmetic expression>

| <additive arithmetic expression>

\* FIRSTS: +, -, AVID\_T, FPL\_T, INL\_T, LPR\_T

\*/

void arithmetic\_expression(){

int code = lookahead\_token.code;

Arr\_Op attrib = lookahead\_token.attribute.arr\_op;

/\*unary arithmetic expression\*/

if (lookahead\_token.code == ART\_OP\_T && (attrib == PLUS || attrib == MINUS)){

unary\_arithmetic\_expression();

}

/\*additive arithmetic expression\*/

else if (code == AVID\_T || code == FPL\_T || code == INL\_T || code == LPR\_T){

additive\_arithmetic\_expression();

}

/\*Otherwise we have an error\*/

else {

syn\_printe();

return;

}

}

/\*

\* PRODUCTION: <unary arithmetic expression> -> - <primary arithmetic expression>

| + <primary arithmetic expression>

\* FIRSTS: +, -

\*/

void unary\_arithmetic\_expression(){

if (lookahead\_token.attribute.arr\_op == MINUS){

match(ART\_OP\_T, MINUS);

}

else if (lookahead\_token.attribute.arr\_op == PLUS){

match(ART\_OP\_T, PLUS);

}

primary\_arithmetic\_expression();

gen\_incode("PLATY: Unary arithmetic expression parsed");

gen\_incode("PLATY: Arithmetic expression parsed");

}

/\*

\* PRODUCTION: <additive arithmetic expression> -> <multiplicative arithmetic expression><additive arithmetic expression1>

\* FIRSTS: AVID\_T, INL\_T, FPL\_T, LPR\_T

\*/

void additive\_arithmetic\_expression(){

multiplicative\_arithmetic\_expression();

additive\_arithmetic\_expression1();

gen\_incode("PLATY: Arithmetic expression parsed");

}

/\*

\* PRODUCTION: <additive arithmetic expression1> ->

+ <multiplicative arithmetic expression><additive arithmetic expression1>

| - <multiplicative arithmetic expression><additive arithmetic expression1>

| empty

\* FIRSTS: +, -, empty

\*/

void additive\_arithmetic\_expression1(){

/\*Match the two possible cases\*/

if (lookahead\_token.code == ART\_OP\_T){

if (lookahead\_token.attribute.arr\_op == MINUS){

match(ART\_OP\_T, MINUS);

}

else if (lookahead\_token.attribute.arr\_op == PLUS){

match(ART\_OP\_T, PLUS);

}

else {

return;

}

/\*Now do the rest\*/

multiplicative\_arithmetic\_expression();

additive\_arithmetic\_expression1();

gen\_incode("PLATY: Additive arithmetic expression parsed");

}

/\*empty\*/

}

/\*

\* PRODUCTION: <multiplicative arithmetic expression> ->

<primary arithmetic expression><multiplicative arithmetic expression1>

\* FIRSTS: AVID\_T, FPL\_T, INL\_T, LPR\_T

\*/

void multiplicative\_arithmetic\_expression(){

primary\_arithmetic\_expression();

multiplicative\_arithmetic\_expression1();

}

/\*

\* PRODUCTION: <multiplicative arithmetic expression1> ->

\* <primary arithmetic expression><multiplicative arithmetic expression1>

| / <primary arithmetic expression><multiplicative arithmetic expression1>

| empty

\* FIRSTS: \*, /

\*/

void multiplicative\_arithmetic\_expression1(){

/\*Match one of the two possible cases\*/

if (lookahead\_token.code == ART\_OP\_T){

if (lookahead\_token.attribute.arr\_op == MULT){

match(ART\_OP\_T, MULT);

}

else if (lookahead\_token.attribute.arr\_op == DIV){

match(ART\_OP\_T, DIV);

}

else {

return;

}

/\*Now do the rest\*/

primary\_arithmetic\_expression();

multiplicative\_arithmetic\_expression1();

gen\_incode("PLATY: Multiplicative arithmetic expression parsed");

}

/\*empty\*/

}

/\*

\* PRODUCTION: <primary arithmetic expression> ->

AVID\_T | FPL\_T | INL\_T | (<arithmetic expression>)

\* FIRSTS: AVID\_T, FPL\_T, INL\_T, LPR\_T

\*/

void primary\_arithmetic\_expression(){

if (lookahead\_token.code == AVID\_T){

match(AVID\_T, NO\_ATTR);

}

else if (lookahead\_token.code == FPL\_T){

match(FPL\_T, NO\_ATTR);

}

else if (lookahead\_token.code == INL\_T){

match(INL\_T, NO\_ATTR);

}

else if (lookahead\_token.code == LPR\_T){

match(LPR\_T, NO\_ATTR);

arithmetic\_expression();

match(RPR\_T, NO\_ATTR);

}

else {

syn\_printe();

return;

}

gen\_incode("PLATY: Primary arithmetic expression parsed");

}

/\*

\* PRODUCTION: <string expression> ->

<primary string expression><string expression1>

\* FIRSTS: SVID\_T, STR\_T

\*/

void string\_expression(){

primary\_string\_expression();

string\_expression1();

gen\_incode("PLATY: String expression parsed");

}

/\*

\* PRODUCTION: <string expression1> ->

<< <primary string expression><string expression1> | empty

\* FIRSTS: SCC\_OP\_T

\*/

void string\_expression1(){

/\*String concatenation\*/

if (lookahead\_token.code == SCC\_OP\_T){

match(SCC\_OP\_T, NO\_ATTR);

primary\_string\_expression();

string\_expression1();

}

/\*do nothing for empty\*/

}

/\*

\* PRODUCTION: <primary string expression> -> SVID\_T | STR\_T

\* FIRSTS: SVID\_T, STR\_T

\*/

void primary\_string\_expression(){

/\*SVID\*/

if (lookahead\_token.code == SVID\_T){

match(SVID\_T, NO\_ATTR);

}

/\*STRING LITERAL\*/

else if (lookahead\_token.code == STR\_T){

match(STR\_T, NO\_ATTR);

}

/\*NO MATCH\*/

else {

syn\_printe();

return;

}

gen\_incode("PLATY: Primary string expression parsed");

}

/\*

\* PRODUCTION: <conditional expression> -> <logical OR expression>

\* FIRSTS: AVID\_T, FPL\_T, INL\_T

\*/

void conditional\_expression(){

logical\_or\_expression();

gen\_incode("PLATY: Conditional expression parsed");

}

/\*

\* PRODUCTION: <logical or expression> -> <logical AND expression> <logical OR expression1>

\* FIRSTS: AVID\_T, FPL\_T, INL\_T

\*/

void logical\_or\_expression(){

logical\_and\_expression();

logical\_or\_expression1();

}

/\*

\* PRODUCTION: <logical or expression1> ->

.OR. <logical AND expression> <logical OR expression1> | empty

\* FIRSTS: .OR., empty

\*/

void logical\_or\_expression1(){

/\*OR\*/

if (lookahead\_token.code == LOG\_OP\_T){

if (lookahead\_token.attribute.log\_op == OR) {

match(LOG\_OP\_T, OR);

}

else if (lookahead\_token.attribute.log\_op == AND){

}

else {

return;

}

/\*Do common things\*/

logical\_and\_expression();

logical\_or\_expression1();

gen\_incode("PLATY: Logical OR expression parsed");

}

/\*Do nothing for empty\*/

}

/\*

\* PRODUCTION: <logical AND expression> -> <relational expression> <logical AND expression1>

\* FIRSTS: AVID\_T, INL\_T, FPL\_T

\*/

void logical\_and\_expression(){

relational\_expression();

logical\_and\_expression1();

}

/\*

\* PRODUCTION: <logical AND expression1> ->

.AND. <relational expression> <logical OR expression1> |empty

\* FIRSTS: .AND., empty

\*/

void logical\_and\_expression1(){

/\*AND\*/

if (lookahead\_token.code == LOG\_OP\_T && lookahead\_token.attribute.log\_op == AND){

match(LOG\_OP\_T, AND);

relational\_expression();

logical\_and\_expression1();

gen\_incode("PLATY: Logical AND expression parsed");

}

/\*empty\*/

}

/\*

\* PRODUCTION: <relational expression> ->

<primary a\_relational expression><primary a\_relational expression1>

| <primary s\_relational expression><primary s\_relational expression1>

\* FIRSTS: AVID\_T, INL\_T, FPL\_T

\*/

void relational\_expression(){

int code = lookahead\_token.code;

/\* A\_RELATIONAL\*/

if (code == AVID\_T || code == FPL\_T || code == INL\_T){

primary\_a\_relational\_expression();

primary\_a\_relational\_expression1();

}

/\* S\_RELATIONAL\*/

else if (code == SVID\_T || code == STR\_T){

primary\_s\_relational\_expression();

primary\_s\_relational\_expression1();

}

/\*INVALID\*/

else {

syn\_printe();

}

gen\_incode("PLATY: Relational expression parsed");

}

/\*

\* PRODUCTION: <primary a\_relational expression> -> AVID\_T | FPL\_T | INL\_T

\* FIRSTS: AVID\_T, FPL\_T, INL\_T

\*/

void primary\_a\_relational\_expression(){

/\*AVID\*/

if (lookahead\_token.code == AVID\_T){

match(AVID\_T, NO\_ATTR);

}

/\*FLOATING POINT LITERAL\*/

else if (lookahead\_token.code == FPL\_T){

match(FPL\_T, NO\_ATTR);

}

/\*INTEGER LITERAL\*/

else if (lookahead\_token.code == INL\_T){

match(INL\_T, NO\_ATTR);

}

/\*Nothing else allowed\*/

else {

syn\_printe();

}

gen\_incode("PLATY: Primary a\_relational expression parsed");

}

/\*

\* PRODUCTION: <primary a\_relational expression1> ->

== <primary a\_relational expression>

| != <primary a\_relational expression>

| > <primary a\_relational expression>

| < <primary a\_relational expression>

\* FIRSTS: ==, !=, >, <

\*/

void primary\_a\_relational\_expression1(){

Rel\_Op attrib = lookahead\_token.attribute.rel\_op;

/\*error\*/

if (lookahead\_token.code != REL\_OP\_T){

syn\_printe();

return;

}

/\*IS EQUAL TO\*/

if (attrib == EQ){

match(REL\_OP\_T, EQ);

}

/\*NOT EQUAL\*/

if (attrib == NE){

match(REL\_OP\_T, NE);

}

/\*GREATER THAN\*/

if (attrib == GT){

match(REL\_OP\_T, GT);

}

/\*LESS THAN\*/

if (attrib == LT){

match(REL\_OP\_T, LT);

}

primary\_a\_relational\_expression();

}

/\*

\* PRODUCTION: <primary s\_relational expression> -> <primary string expression>

\* FIRSTS: ==, !=, >, <

\*/

void primary\_s\_relational\_expression(){

primary\_string\_expression();

gen\_incode("PLATY: Primary s\_relational expression parsed");

}

/\*

\*PRODUCTION: <primary s\_relational expression1> ->

== <primary s\_relational expression>

| != <primary s\_relational expression>

| > <primary s\_relational expression>

| < <primary s\_relational expression>

\* FIRSTS: ==, !=, >, <

\*/

void primary\_s\_relational\_expression1(){

Rel\_Op attrib = lookahead\_token.attribute.rel\_op;

/\*error\*/

if (lookahead\_token.code != REL\_OP\_T){

syn\_printe();

return;

}

/\*IS EQUAL TO\*/

if (attrib == EQ){

match(REL\_OP\_T, EQ);

}

/\*NOT EQUAL\*/

if (attrib == NE){

match(REL\_OP\_T, NE);

}

/\*GREATER THAN\*/

if (attrib == GT){

match(REL\_OP\_T, GT);

}

/\*LESS THAN\*/

if (attrib == LT){

match(REL\_OP\_T, LT);

}

primary\_s\_relational\_expression();

}

Parsing the source file...

PLATY: Primary arithmetic expression parsed

PLATY: Unary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Variable list parsed

PLATY: INPUT statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Logical OR expression parsed

PLATY: Conditional expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: USING statement parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Primary string expression parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: Primary string expression parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: IF statement parsed

PLATY: Output list (string literal) parsed

PLATY: OUTPUT statement parsed

PLATY: Variable list parsed

PLATY: OUTPUT statement parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Logical AND expression parsed

PLATY: Logical OR expression parsed

PLATY: Conditional expression parsed

PLATY: Variable list parsed

PLATY: OUTPUT statement parsed

PLATY: Output list (empty) parsed

PLATY: OUTPUT statement parsed

PLATY: Opt\_statements parsed

PLATY: IF statement parsed

PLATY: Program parsed

PLATY: Source file parsed

Collecting garbage...

Reading file ass4e.pls ....Please wait

Printing input buffer parameters:

The capacity of the buffer is: 152

The current size of the buffer is: 151

Printing input buffer contents:

!<This is the greatest of all programs

!<The law of computer programming:

!<"If a program is useless, it will have to be well documented."

PLATYPUS{}

Parsing the source file...

PLATY: Opt\_statements parsed

PLATY: Program parsed

PLATY: Source file parsed

Collecting garbage...

Reading file ass4r.pls ....Please wait

Printing input buffer parameters:

The capacity of the buffer is: 451

The current size of the buffer is: 450

Printing input buffer contents:

!<This is a syntactically correct PLATYPUS program

!<Weiler's law:

!<"Nothing is impossible for the man who doesn't have to do it himself."

!<"Parsing is passing." S^R & Compilers' law

PLATYPUS{

a=-01;

b=+0.;

INPUT(c);

INPUT(d,e,f);

c=((d+e)/a)\*f-(((d-e)\*a)/f);

USING(a = a,a!=b .OR. c==d .AND. e<f .OR. a>0,c = e)REPEAT{

IF(a==1 .AND. b==0.0)THEN

c=-(5.9);

ELSE {c=-c;};

};

OUTPUT();

OUTPUT("Results: ");

OUTPUT(d,e,f,a);

}

Parsing the source file...

PLATY: Primary arithmetic expression parsed

PLATY: Unary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Unary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Variable list parsed

PLATY: INPUT statement parsed

PLATY: Variable list parsed

PLATY: INPUT statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Logical AND expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Logical OR expression parsed

PLATY: Logical OR expression parsed

PLATY: Conditional expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Logical AND expression parsed

PLATY: Conditional expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Unary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Unary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: IF statement parsed

PLATY: USING statement parsed

PLATY: Output list (empty) parsed

PLATY: OUTPUT statement parsed

PLATY: Output list (string literal) parsed

PLATY: OUTPUT statement parsed

PLATY: Variable list parsed

PLATY: OUTPUT statement parsed

PLATY: Program parsed

PLATY: Source file parsed

Collecting garbage...

Reading file ass4w1.pls ....Please wait

Printing input buffer parameters:

The capacity of the buffer is: 286

The current size of the buffer is: 285

Printing input buffer contents:

!<This program contains a syntax error

PLATYPUS{

OUTPUT("If there is a possibility of several things going wrong,

the one that will cause the most damage will be the first

one to go wrong. Murphy's law");

c=(-(d+e))\*f-

((d-e)\*a)/f); !< Missing "(" or extra ")"

}

Parsing the source file...

PLATY: Output list (string literal) parsed

PLATY: OUTPUT statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Unary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Syntax error: Line: 8

\*\*\*\*\* Token code: 13 Attribute: NA

PLATY: Assignment statement parsed

PLATY: Program parsed

PLATY: Source file parsed

Syntax errors: 1

Collecting garbage...

Reading file ass4w2.pls ....Please wait

Printing input buffer parameters:

The capacity of the buffer is: 171

The current size of the buffer is: 170

Printing input buffer contents:

!<This program contains syntax errors

!<Ginsberg's theorems:

!<1.You can't win

!<2.You can't break even

!<3.You can't even quit the game

PLATYPUS{

a\_ = a + . (- b);

};

Parsing the source file...

PLATY: Syntax error: Line: 7

\*\*\*\*\* Token code: 0 Attribute: \_

PLATY: Primary arithmetic expression parsed

PLATY: Syntax error: Line: 7

\*\*\*\*\* Token code: 0 Attribute: .

PLATY: Primary arithmetic expression parsed

PLATY: Unary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Program parsed

PLATY: Syntax error: Line: 8

\*\*\*\*\* Token code: 18 Attribute: NA

PLATY: Source file parsed

Syntax errors: 3

Collecting garbage...

Reading file ass4w3.pls ....Please wait

Printing input buffer parameters:

The capacity of the buffer is: 351

The current size of the buffer is: 350

Printing input buffer contents:

!<This program contains a syntax error

!<Forsyth's second corollary to Murphy's law:

!<"Just when you see the light at the end of the tunnel, the roof caves."

PLATYPUS{

INPUT(IF);

a# = "";

INPUT(b,a,b#);

IF(if == "begin" .AND a# |= a)THEN

a# = "begin" << a# << b# << "end";

^@#$& ELSE{REPEAT};

PLATYPUS !<What is this creature doing here?

}

Parsing the source file...

PLATY: Syntax error: Line: 6

\*\*\*\*\* Token code: 16 Attribute: IF

PLATY: Variable list parsed

PLATY: Syntax error: Line: 6

\*\*\*\*\* Token code: 16 Attribute: IF

PLATY: INPUT statement parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: Variable list parsed

PLATY: INPUT statement parsed

PLATY: Primary a\_relational expression parsed

PLATY: Syntax error: Line: 9

\*\*\*\*\* Token code: 6 Attribute: begin

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Syntax error: Line: 9

\*\*\*\*\* Token code: 6 Attribute: begin

PLATY: Primary string expression parsed

PLATY: Primary string expression parsed

PLATY: Primary string expression parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Syntax error: Line: 12

\*\*\*\*\* Token code: 0 Attribute: ^

PLATY: Assignment statement parsed

PLATY: Syntax error: Line: 12

\*\*\*\*\* Token code: 0 Attribute: @

PLATY: Opt\_statements parsed

PLATY: Syntax error: Line: 12

\*\*\*\*\* Token code: 16 Attribute: REPEAT

PLATY: IF statement parsed

PLATY: Syntax error: Line: 13

\*\*\*\*\* Token code: 16 Attribute: PLATYPUS

PLATY: Program parsed

PLATY: Source file parsed

Syntax errors: 6

Collecting garbage...

Reading file ass4w4.pls ....Please wait

Printing input buffer parameters:

The capacity of the buffer is: 418

The current size of the buffer is: 417

Printing input buffer contents:

!<This program contains syntax errors

!<Addition to Murphy's laws:

!<"In nature, nothing is ever right. Therefore,

!<if everything is going right ... something is wrong."

PLATYPUS{

USING(a=a,a == b \* c,c=c)REPEAT{

IF()THEN a=b!=c; ELSE ELSE{};

USING(a=a,good# != "a",)REPEAT{};

a = (a badLong89#) + (good badLong89) \* (good 7.);

};

IF (c > 0)THEN ELSE{};

OUTPUT("Imagine all the scanners );

}

}

Parsing the source file...

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Syntax error: Line: 7

\*\*\*\*\* Token code: 9 Attribute: 2

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Syntax error: Line: 8

\*\*\*\*\* Token code: 13 Attribute: NA

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Syntax error: Line: 8

\*\*\*\*\* Token code: 10 Attribute: 1

PLATY: Assignment statement parsed

PLATY: Syntax error: Line: 8

\*\*\*\*\* Token code: 16 Attribute: ELSE

PLATY: Opt\_statements parsed

PLATY: IF statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Syntax error: Line: 9

\*\*\*\*\* Token code: 13 Attribute: NA

PLATY: Opt\_statements parsed

PLATY: USING statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Syntax error: Line: 10

\*\*\*\*\* Token code: 3 Attribute: badLong#

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Syntax error: Line: 10

\*\*\*\*\* Token code: 2 Attribute: badLong8

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Syntax error: Line: 10

\*\*\*\*\* Token code: 4 Attribute: 7.0

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: USING statement parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Opt\_statements parsed

PLATY: Opt\_statements parsed

PLATY: IF statement parsed

PLATY: Syntax error: Line: 15

\*\*\*\*\* Token code: 0 Attribute: "Imagine all the ...

PLATY: Output list (empty) parsed

PLATY: Syntax error: Line: 15

\*\*\*\*\* Token code: 1 Attribute: NA

Syntax errors: 8

Collecting garbage...

Reading file correct.pls ....Please wait

Printing input buffer parameters:

The capacity of the buffer is: 1180

The current size of the buffer is: 1179

Printing input buffer contents:

!<This file will test the parser for full informal grammar specification

!<This should be regarded as an error free program

PLATYPUS {

!<Initialized variable declarations

iInit = 1;

jInit = 2;

mInit = 010; !<The number 8 in octal

nInit = 4;

float = 1.0;

anotherFloat = 20.;

iVar = iInit;

jVar = jInit;

!<Switch data type

mVar = 2.0;

nVar = 0.010;

!<String variables

iString# = "Pizza";

aString# = "Fallsview";

!< Some arithmetic statements

iInit = (iInit + anotherFloat - 3) \* 012 /2.3; !<Mix of integer and floating point

iString#= iString# << aString# << "Billy Jean";

!<Selection Statement

IF(iString# == aString#) THEN

iString# = "GOODLY";

ELSE {

iString# = "BADLY";

};

!<Optional Statements

IF (iString# == "BADLY") THEN

ELSE { };

!<Iteration Statement

USING(i = 0, i < jInit, i = i + 1) REPEAT {

aString# = aString# << "i";

};

!<Optional iteration

USING(i = 0, i < jInit .AND. jInit != 50 .OR. iGo == 1, i = i + 1) REPEAT { };

!<Input and output statements

INPUT(iInput1);

OUTPUT(iInput1);

INPUT(iInput2);

OUTPUT(iInput2);

INPUT(iInput1, iInput2);

OUTPUT();

OUTPUT(iInput1, iInput2);

OUTPUT("BINGO");

}

Parsing the source file...

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Multiplicative arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Assignment statement parsed

PLATY: Primary string expression parsed

PLATY: Primary string expression parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: IF statement parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Primary string expression parsed

PLATY: Primary s\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Opt\_statements parsed

PLATY: Opt\_statements parsed

PLATY: IF statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Conditional expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary string expression parsed

PLATY: Primary string expression parsed

PLATY: String expression parsed

PLATY: Assignment expression (string) parsed

PLATY: Assignment statement parsed

PLATY: USING statement parsed

PLATY: Primary arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Logical AND expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Primary a\_relational expression parsed

PLATY: Relational expression parsed

PLATY: Logical OR expression parsed

PLATY: Conditional expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Primary arithmetic expression parsed

PLATY: Additive arithmetic expression parsed

PLATY: Arithmetic expression parsed

PLATY: Assignment expression (arithmetic) parsed

PLATY: Opt\_statements parsed

PLATY: USING statement parsed

PLATY: Variable list parsed

PLATY: INPUT statement parsed

PLATY: Variable list parsed

PLATY: OUTPUT statement parsed

PLATY: Variable list parsed

PLATY: INPUT statement parsed

PLATY: Variable list parsed

PLATY: OUTPUT statement parsed

PLATY: Variable list parsed

PLATY: INPUT statement parsed

PLATY: Output list (empty) parsed

PLATY: OUTPUT statement parsed

PLATY: Variable list parsed

PLATY: OUTPUT statement parsed

PLATY: Output list (string literal) parsed

PLATY: OUTPUT statement parsed

PLATY: Program parsed

PLATY: Source file parsed

Collecting garbage...

**TEST PLAN**

As usual, testing required that I match your test files. I created a script that would create and test each of your test files. It looked like this:

ass4 ass4e.pls > ass4e.out

ass4 ass4r.pls > ass4r.out

ass4 ass4w1.pls > ass4w1.out

ass4 ass4w2.pls > ass4w2.out

ass4 ass4w3.pls > ass4w3.out

ass4 ass4w4.pls > ass4w4.out

ass4 ass2r.pls > ass2r.out

diff ass4e.out ../Assignment4MPTF/ass4e.out

diff ass4r.out ../Assignment4MPTF/ass4r.out

diff ass4w1.out ../Assignment4MPTF/ass4w1.out

diff ass4w2.out ../Assignment4MPTF/ass4w2.out

diff ass4w3.out ../Assignment4MPTF/ass4w3.out

diff ass4w4.out ../Assignment4MPTF/ass4w4.out

diff ass2r.out ../AddParserTest/ass2r.out

pause

I have also created additional test files to ensure functionality, they have been included in my zip file, and the output has been included both in my zip file, and in this document.

**BONUS 3**

I have chosen to do bonus 3, the interpreter. I guarantee that you will not like it. It is very ad hoc, and put together in a very messy way. But for the most part it is a working interpreter. Throughout most of the code you will see stacks sent into many of the production functions. These stacks are what I used to implement things like arithmetic and string expressions, relational expressions. Almost everything uses stacks in my implementation. One thing to note is that I breached the encapsulation of the scanner, making “forward” a global variable that I could modify. This allowed me to pull the scanner back and recycle through tokens when implementing the USING statement. Errors are shown in the same way as they had originally in assignment 4, I did not have time to change it.

For testing, I have a few .pls files included that are named according to what they are testing, and also one file named tictactoe.pls. I stole this source program from one of the hall of fame cross-compiler students, the first one in the list. I’m too lazy to go find the name. The error handling was checked in the parser, so my test files will only contain syntactically correct programs. My goal with them is to test the real functionality of the interpreter, not the parser. I spent a good deal of time on this assignment, and I hope my code doesn’t make you too angry ☺

Thanks for the wonderful semester.

#V