Utsab Khakurel

@02752480

Github Link: <https://github.com/Utsabab/Syntax-Analysis-Programming-Assignment->

commit 76d4e75348bd97470d02150f433f7e9c7403389b

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Thu Mar 23 23:23:00 2017 -0400

Error function end

diff --git a/front.c b/front.c

index 31935b3..e5d1506 100644

--- a/front.c

+++ b/front.c

@@ -304,19 +304,25 @@ parenthesis \*/

expr();

if (nextToken == RIGHT\_PAREN)

lex();

- else

+ else {

error();

+ return;

+ }

} /\* End of if (nextToken == ... \*/

/\* It was not an id, an integer literal, or a left

parenthesis \*/

- else

+ else {

error();

+ return;

+ }

} /\* End of else \*/

printf("Exit <factor>\n");;

}

/\* End of function factor \*/

void error () {

-

+ printf("Line %d: %d: %s:", num\_row, num\_col, each\_line);

+ printf("Syntax error: %s\n Error occurs at %s\n", read\_till, last\_read\_lex);

+ break\_at = 1;

}

\ No newline at end of file

diff --git a/parser b/parser

new file mode 100755

index 0000000..3f6a590

Binary files /dev/null and b/parser differ

commit 834dd4aed87ed344701df4eb9a489aa53c4638a0

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Thu Mar 23 23:14:48 2017 -0400

creating error function

diff --git a/front.c b/front.c

index 23444c4..31935b3 100644

--- a/front.c

+++ b/front.c

@@ -315,4 +315,8 @@ parenthesis \*/

} /\* End of else \*/

printf("Exit <factor>\n");;

}

-/\* End of function factor \*/

\ No newline at end of file

+/\* End of function factor \*/

+

+void error () {

+

+}

\ No newline at end of file

commit bf28a6b993946c62ed56f80a38a718a8e6813e32

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Thu Mar 23 23:09:56 2017 -0400

Similar changes in term function as in expr function to handle error

diff --git a/front.c b/front.c

index 61f1d80..23444c4 100644

--- a/front.c

+++ b/front.c

@@ -247,7 +247,7 @@ void expr() {

printf("Enter <expr>\n");

/\* Parse the first term \*/

term();

- if (break\_at = 1) {

+ if (break\_at == 1) {

return;

}

/\* As long as the next token is + or -, get

@@ -270,11 +270,17 @@ void term() {

printf("Enter <term>\n");

/\* Parse the first factor \*/

factor();

+ if (break\_at == 1) {

+ return;

+ }

/\* As long as the next token is \* or /, get the

next token and parse the next factor \*/

while (nextToken == MULT\_OP || nextToken == DIV\_OP) {

lex();

factor();

+ if (break\_at == 1) {

+ return;

+ }

}

printf("Exit <term>\n");

} /\* End of function term \*/

commit a87ba6a104cab1a69b359519ff61ad2ffbdb302a

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Thu Mar 23 23:02:57 2017 -0400

Changes in the expr function to handle the error

diff --git a/front.c b/front.c

index d8ce2ed..61f1d80 100644

--- a/front.c

+++ b/front.c

@@ -23,8 +23,9 @@ int num\_col = 1;

int num\_row = 0;

int break\_at = 0;

int char\_index;

+char last\_read\_char;

char read\_till[1000];

-char last\_read[1000];

+char last\_read\_lex[1000];

/\* Function declarations \*/

void addChar();

@@ -70,7 +71,7 @@ void main(int argc, char\* argv[]) {

else {

while ((read = getline(&each\_line, &len, in\_fp)) != -1) {

strcpy(read\_till, "");

- strcpy(last\_read, "");

+ strcpy(last\_read\_lex, "");

num\_row++;

num\_col = 1;

char\_index = 0;

@@ -186,8 +187,8 @@ void getNonBlank() {

int lex() {

lexLen = 0;

getNonBlank();

- last\_read = nextChar;

- strcpy(last\_read, lexeme);

+ last\_read\_char = nextChar;

+ strcpy(last\_read\_lex, lexeme);

switch (charClass) {

/\* Parse identifiers \*/

@@ -198,8 +199,8 @@ int lex() {

addChar();

getChar();

}

- strcat(curr\_read, lexeme);

- strcat(curr\_read, " ");

+ strcat(read\_till, lexeme);

+ strcat(read\_till, " ");

nextToken = IDENT;

break;

@@ -211,8 +212,8 @@ int lex() {

addChar();

getChar();

}

- strcat(curr\_read, lexeme);

- strcat(curr\_read, " ");

+ strcat(read\_till, lexeme);

+ strcat(read\_till, " ");

nextToken = INT\_LIT;

break;

@@ -246,11 +247,17 @@ void expr() {

printf("Enter <expr>\n");

/\* Parse the first term \*/

term();

+ if (break\_at = 1) {

+ return;

+ }

/\* As long as the next token is + or -, get

the next token and parse the next term \*/

while (nextToken == ADD\_OP || nextToken == SUB\_OP) {

lex();

term();

+ if (break\_at == 1) {

+ return;

+ }

}

printf("Exit <expr>\n");

} /\* End of function expr \*/

commit bbdf6d53d015df5079051368a2f72e904a742bfa

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Thu Mar 23 22:53:42 2017 -0400

Used while loop in the main function to read multiple input lines and fixes in other function

diff --git a/front.c b/front.c

index fe80bf3..d8ce2ed 100644

--- a/front.c

+++ b/front.c

@@ -3,6 +3,8 @@

#include <stdio.h>

#include <ctype.h>

+#include <string.h>

+#include <stdlib.h>

/\* Global declarations \*/

/\* Variables \*/

@@ -17,6 +19,12 @@ FILE \*in\_fp, \*fopen();

char\* each\_line = NULL;

size\_t read = 0;

size\_t len = 0;

+int num\_col = 1;

+int num\_row = 0;

+int break\_at = 0;

+int char\_index;

+char read\_till[1000];

+char last\_read[1000];

/\* Function declarations \*/

void addChar();

@@ -61,15 +69,25 @@ void main(int argc, char\* argv[]) {

printf("ERROR - cannot open %s\n", file);

else {

while ((read = getline(&each\_line, &len, in\_fp)) != -1) {

-

- }

- getChar();

- do {

- lex();

- expr();

- } while (nextToken != EOF);

+ strcpy(read\_till, "");

+ strcpy(last\_read, "");

+ num\_row++;

+ num\_col = 1;

+ char\_index = 0;

+ break\_at = 0;

+ getChar();

+ if (each\_line != NULL) {

+ do {

+ lex();

+ expr();

+ if (break\_at == 1) {

+ break;

+ }

+ } while (nextToken != EOF);

+

+ }

+ }

}

- fclose(in\_fp);

}

//while (fgets()) {

@@ -116,7 +134,9 @@ int lookup(char ch) {

nextToken = EOF;

break;

}

- return nextToken;

+ strcat(read\_till, lexeme);

+ strcat(read\_till, " ");

+ return nextToken;

}

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

@@ -136,7 +156,9 @@ void addChar() {

input and determine its character class \*/

void getChar() {

- if ((nextChar = getc(in\_fp)) != EOF) {

+ if (each\_line[char\_index] != '\n' && each\_line[char\_index] != '\0') {

+ num\_col++;

+ nextChar = each\_line[char\_index++];

if (isalpha(nextChar))

charClass = LETTER;

else if (isdigit(nextChar))

@@ -164,6 +186,8 @@ void getNonBlank() {

int lex() {

lexLen = 0;

getNonBlank();

+ last\_read = nextChar;

+ strcpy(last\_read, lexeme);

switch (charClass) {

/\* Parse identifiers \*/

@@ -174,6 +198,8 @@ int lex() {

addChar();

getChar();

}

+ strcat(curr\_read, lexeme);

+ strcat(curr\_read, " ");

nextToken = IDENT;

break;

@@ -185,6 +211,8 @@ int lex() {

addChar();

getChar();

}

+ strcat(curr\_read, lexeme);

+ strcat(curr\_read, " ");

nextToken = INT\_LIT;

break;

diff --git a/front.in b/front.in

index 283a904..664bf14 100644

--- a/front.in

+++ b/front.in

@@ -1 +1,4 @@

-(sum + 47) / total

\ No newline at end of file

+(sum + 47) / total

+a + b

+12 - 7 / 5

+(temp - 2) / 10

\ No newline at end of file

commit b41a55d6c1633b217f9682a43d5e00d46963a5ba

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Thu Mar 23 22:13:22 2017 -0400

Working on main function to read all lines of input

diff --git a/front.c b/front.c

index 2cb0a85..fe80bf3 100644

--- a/front.c

+++ b/front.c

@@ -14,6 +14,10 @@ int token;

int nextToken;

FILE \*in\_fp, \*fopen();

+char\* each\_line = NULL;

+size\_t read = 0;

+size\_t len = 0;

+

/\* Function declarations \*/

void addChar();

void getChar();

@@ -43,12 +47,22 @@ int lex();

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

/\* main driver \*/

-main() {

+void main(int argc, char\* argv[]) {

+

+ if (argc != 2) {

+ printf("Number of argument doesn't match");

+ exit(0);

+ }

/\* Open the input data file and process its contents \*/

- if ((in\_fp = fopen("front.in", "r")) == NULL)

- printf("ERROR - cannot open front.in \n");

+ char\* file = argv[1];

+

+ if ((in\_fp = fopen(file, "r")) == NULL)

+ printf("ERROR - cannot open %s\n", file);

else {

+ while ((read = getline(&each\_line, &len, in\_fp)) != -1) {

+

+ }

getChar();

do {

lex();

@@ -58,9 +72,9 @@ main() {

fclose(in\_fp);

}

-while (fgets()) {

+//while (fgets()) {

+

-}

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

/\* lookup - a function to lookup operators and parentheses

and return the token \*/

@@ -82,25 +96,25 @@ int lookup(char ch) {

nextToken = ADD\_OP;

break;

- case '-':

- addChar();

- nextToken = SUB\_OP;

- break;

-

- case '\*':

- addChar();

- nextToken = MULT\_OP;

- break;

-

- case '/':

- addChar();

- nextToken = DIV\_OP;

- break;

-

- default:

- addChar();

- nextToken = EOF;

- break;

+ case '-':

+ addChar();

+ nextToken = SUB\_OP;

+ break;

+

+ case '\*':

+ addChar();

+ nextToken = MULT\_OP;

+ break;

+

+ case '/':

+ addChar();

+ nextToken = DIV\_OP;

+ break;

+

+ default:

+ addChar();

+ nextToken = EOF;

+ break;

}

return nextToken;

}

diff --git a/front.in b/front.in

index 2fdfd51..283a904 100644

--- a/front.in

+++ b/front.in

@@ -1 +1 @@

-a + b

\ No newline at end of file

+(sum + 47) / total

\ No newline at end of file

commit 7823dac98552302ed663265d10f51eb3c13dd800

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Thu Mar 23 02:34:05 2017 -0400

Indentation and slight error fixes

diff --git a/front.c b/front.c

index 2560079..2cb0a85 100644

--- a/front.c

+++ b/front.c

@@ -46,72 +46,75 @@ int lex();

main() {

/\* Open the input data file and process its contents \*/

- if ((in\_fp = fopen("front.in", "r")) == NULL)

- printf("ERROR - cannot open front.in \n");

- else {

- getChar();

- do {

- lex();

- expr();

- } while (nextToken != EOF);

- }

- fclose(in\_fp);

+ if ((in\_fp = fopen("front.in", "r")) == NULL)

+ printf("ERROR - cannot open front.in \n");

+ else {

+ getChar();

+ do {

+ lex();

+ expr();

+ } while (nextToken != EOF);

+ }

+ fclose(in\_fp);

}

+while (fgets()) {

+

+}

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

/\* lookup - a function to lookup operators and parentheses

and return the token \*/

int lookup(char ch) {

- switch (ch) {

- case '(':

- addChar();

- nextToken = LEFT\_PAREN;

- break;

-

- case ')':

- addChar();

- nextToken = RIGHT\_PAREN;

- break;

-

- case '+':

- addChar();

- nextToken = ADD\_OP;

- break;

-

- case '-':

- addChar();

- nextToken = SUB\_OP;

- break;

-

- case '\*':

- addChar();

- nextToken = MULT\_OP;

- break;

-

- case '/':

- addChar();

- nextToken = DIV\_OP;

- break;

-

- default:

- addChar();

- nextToken = EOF;

- break;

- }

- return nextToken;

+ switch (ch) {

+ case '(':

+ addChar();

+ nextToken = LEFT\_PAREN;

+ break;

+

+ case ')':

+ addChar();

+ nextToken = RIGHT\_PAREN;

+ break;

+

+ case '+':

+ addChar();

+ nextToken = ADD\_OP;

+ break;

+

+ case '-':

+ addChar();

+ nextToken = SUB\_OP;

+ break;

+

+ case '\*':

+ addChar();

+ nextToken = MULT\_OP;

+ break;

+

+ case '/':

+ addChar();

+ nextToken = DIV\_OP;

+ break;

+

+ default:

+ addChar();

+ nextToken = EOF;

+ break;

+ }

+ return nextToken;

}

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

/\* addChar - a function to add nextChar to lexeme \*/

void addChar() {

- if (lexLen <= 98) {

- lexeme[lexLen++] = nextChar;

- lexeme[lexLen] = 0;

- }

- else

- printf("Error - lexeme is too long \n");

+ if (lexLen <= 98) {

+ lexeme[lexLen++] = nextChar;

+ lexeme[lexLen] = 0;

+ }

+ else

+ printf("Error - lexeme is too long \n");

}

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

@@ -119,15 +122,15 @@ void addChar() {

input and determine its character class \*/

void getChar() {

- if ((nextChar = getc(in\_fp)) != EOF) {

- if (isalpha(nextChar))

- charClass = LETTER;

- else if (isdigit(nextChar))

- charClass = DIGIT;

- else charClass = UNKNOWN;

- }

- else

- charClass = EOF;

+ if ((nextChar = getc(in\_fp)) != EOF) {

+ if (isalpha(nextChar))

+ charClass = LETTER;

+ else if (isdigit(nextChar))

+ charClass = DIGIT;

+ else charClass = UNKNOWN;

+ }

+ else

+ charClass = EOF;

}

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

@@ -135,8 +138,8 @@ void getChar() {

returns a non-whitespace character \*/

void getNonBlank() {

- while (isspace(nextChar))

- getChar();

+ while (isspace(nextChar))

+ getChar();

}

@@ -145,51 +148,52 @@ void getNonBlank() {

expressions \*/

int lex() {

- lexLen = 0;

- getNonBlank();

- switch (charClass) {

-

- /\* Parse identifiers \*/

- case LETTER:

- addChar();

- getChar();

- while (charClass == LETTER || charClass == DIGIT) {

- addChar();

- getChar();

- }

- nextToken = IDENT;

- break;

-

-/\* Parse integer literals \*/

- case DIGIT:

- addChar();

- getChar();

- while (charClass == DIGIT) {

- addChar();

- getChar();

- }

- nextToken = INT\_LIT;

- break;

-

-/\* Parentheses and operators \*/

- case UNKNOWN:

- lookup(nextChar);

- getChar();

- break;

-

-/\* EOF \*/

- case EOF:

- nextToken = EOF;

- lexeme[0] = 'E';

- lexeme[1] = 'O';

- lexeme[2] = 'F';

- lexeme[3] = 0;

- break;

-

- } /\* End of switch \*/

- printf("Next token is: %d, Next lexeme is %s\n",

- nextToken, lexeme);

- return nextToken;

+ lexLen = 0;

+ getNonBlank();

+ switch (charClass) {

+

+ /\* Parse identifiers \*/

+ case LETTER:

+ addChar();

+ getChar();

+ while (charClass == LETTER || charClass == DIGIT) {

+ addChar();

+ getChar();

+ }

+ nextToken = IDENT;

+ break;

+

+ /\* Parse integer literals \*/

+ case DIGIT:

+ addChar();

+ getChar();

+ while (charClass == DIGIT) {

+ addChar();

+ getChar();

+ }

+ nextToken = INT\_LIT;

+ break;

+

+ /\* Parentheses and operators \*/

+ case UNKNOWN:

+ lookup(nextChar);

+ getChar();

+ break;

+

+ /\* EOF \*/

+ case EOF:

+ nextToken = EOF;

+ lexeme[0] = 'E';

+ lexeme[1] = 'O';

+ lexeme[2] = 'F';

+ lexeme[3] = 0;

+ break;

+

+ } /\* End of switch \*/

+

+ printf("Next token is: %d, Next lexeme is %s\n",

+ nextToken, lexeme);

+ return nextToken;

} /\* End of function lex \*/

/\* expr

@@ -197,16 +201,16 @@ Parses strings in the language generated by the rule:

<expr> -> <term> {(+ | -) <term>}

\*/

void expr() {

-printf("Enter <expr>\n");

+ printf("Enter <expr>\n");

/\* Parse the first term \*/

-term();

+ term();

/\* As long as the next token is + or -, get

the next token and parse the next term \*/

-while (nextToken == ADD\_OP || nextToken == SUB\_OP) {

-lex();

-term();

-}

-printf("Exit <expr>\n");

+ while (nextToken == ADD\_OP || nextToken == SUB\_OP) {

+ lex();

+ term();

+ }

+ printf("Exit <expr>\n");

} /\* End of function expr \*/

/\* term

@@ -214,16 +218,16 @@ Parses strings in the language generated by the rule:

<term> -> <factor> {(\* | /) <factor>)

\*/

void term() {

-printf("Enter <term>\n");

+ printf("Enter <term>\n");

/\* Parse the first factor \*/

-factor();

+ factor();

/\* As long as the next token is \* or /, get the

next token and parse the next factor \*/

-while (nextToken == MULT\_OP || nextToken == DIV\_OP) {

-lex();

-factor();

-}

-printf("Exit <term>\n");

+ while (nextToken == MULT\_OP || nextToken == DIV\_OP) {

+ lex();

+ factor();

+ }

+ printf("Exit <term>\n");

} /\* End of function term \*/

/\* factor

@@ -231,28 +235,29 @@ Parses strings in the language generated by the rule:

<factor> -> id | int\_constant | ( <expr )

\*/

void factor() {

-printf("Enter <factor>\n");

+ printf("Enter <factor>\n");

/\* Determine which RHS \*/

-if (nextToken == IDENT || nextToken == INT\_LIT)

+ if (nextToken == IDENT || nextToken == INT\_LIT)

/\* Get the next token \*/

-lex();

+ lex();

/\* If the RHS is ( <expr>), call lex to pass over the

left parenthesis, call expr, and check for the right

parenthesis \*/

-else {

-if (nextToken == LEFT\_PAREN) {

-lex();

-expr();

-if (nextToken == RIGHT\_PAREN)

-lex();

-else

-error();

-} /\* End of if (nextToken == ... \*/

+ else {

+ if (nextToken == LEFT\_PAREN) {

+ lex();

+ expr();

+ if (nextToken == RIGHT\_PAREN)

+ lex();

+ else

+ error();

+ } /\* End of if (nextToken == ... \*/

+

/\* It was not an id, an integer literal, or a left

parenthesis \*/

-else

-error();

-} /\* End of else \*/

-printf("Exit <factor>\n");;

+ else

+ error();

+ } /\* End of else \*/

+ printf("Exit <factor>\n");;

}

/\* End of function factor \*/

\ No newline at end of file

commit 4d70e6c20a0170adaec7c3bf718634a53a0f4408

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Wed Mar 22 23:27:11 2017 -0400

Added front.in output file, prints out the desired output for one line of input file

diff --git a/a.out b/a.out

index b343b18..a1bad83 100755

Binary files a/a.out and b/a.out differ

diff --git a/front.c b/front.c

index b0d07d9..2560079 100644

--- a/front.c

+++ b/front.c

@@ -18,6 +18,10 @@ FILE \*in\_fp, \*fopen();

void addChar();

void getChar();

void getNonBlank();

+void term();

+void expr();

+void factor();

+void error();

int lex();

/\* Character classes \*/

@@ -47,9 +51,11 @@ main() {

else {

getChar();

do {

- lex();

+ lex();

+ expr();

} while (nextToken != EOF);

}

+ fclose(in\_fp);

}

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

@@ -186,67 +192,67 @@ int lex() {

return nextToken;

} /\* End of function lex \*/

- /\* expr

- Parses strings in the language generated by the rule:

- <expr> -> <term> {(+ | -) <term>} \*/

-

+/\* expr

+Parses strings in the language generated by the rule:

+<expr> -> <term> {(+ | -) <term>}

+\*/

void expr() {

- printf("Enter <expr>\n");

+printf("Enter <expr>\n");

/\* Parse the first term \*/

- term();

+term();

/\* As long as the next token is + or -, get

- the next token and parse the next term \*/

- while (nextToken == ADD\_OP || nextToken == SUB\_OP) {

- lex();

- term();

- }

- printf("Exit <expr>\n");

+the next token and parse the next term \*/

+while (nextToken == ADD\_OP || nextToken == SUB\_OP) {

+lex();

+term();

+}

+printf("Exit <expr>\n");

} /\* End of function expr \*/

- /\* term

- Parses strings in the language generated by the rule:

- <term> -> <factor> {(\* | /) <factor>)

- \*/

+/\* term

+Parses strings in the language generated by the rule:

+<term> -> <factor> {(\* | /) <factor>)

+\*/

void term() {

- printf("Enter <term>\n");

+printf("Enter <term>\n");

/\* Parse the first factor \*/

- factor();

+factor();

/\* As long as the next token is \* or /, get the

- next token and parse the next factor \*/

- while (nextToken == MULT\_OP || nextToken == DIV\_OP) {

- lex();

- factor();

- }

- printf("Exit <term>\n");

+next token and parse the next factor \*/

+while (nextToken == MULT\_OP || nextToken == DIV\_OP) {

+lex();

+factor();

+}

+printf("Exit <term>\n");

} /\* End of function term \*/

- /\* factor

- Parses strings in the language generated by the rule:

- <factor> -> id | int\_constant | ( <expr )

- \*/

+/\* factor

+Parses strings in the language generated by the rule:

+<factor> -> id | int\_constant | ( <expr )

+\*/

void factor() {

- printf("Enter <factor>\n");

+printf("Enter <factor>\n");

/\* Determine which RHS \*/

- if (nextToken == IDENT || nextToken == INT\_LIT)

- /\* Get the next token \*/

- lex();

+if (nextToken == IDENT || nextToken == INT\_LIT)

+/\* Get the next token \*/

+lex();

/\* If the RHS is ( <expr>), call lex to pass over the

- left parenthesis, call expr, and check for the right

- parenthesis \*/

- else {

- if (nextToken == LEFT\_PAREN) {

- lex();

- expr();

- if (nextToken == RIGHT\_PAREN)

- lex();

- else

- error();

- } /\* End of if (nextToken == ... \*/

-

+left parenthesis, call expr, and check for the right

+parenthesis \*/

+else {

+if (nextToken == LEFT\_PAREN) {

+lex();

+expr();

+if (nextToken == RIGHT\_PAREN)

+lex();

+else

+error();

+} /\* End of if (nextToken == ... \*/

/\* It was not an id, an integer literal, or a left

- parenthesis \*/

- else

- error();

- } /\* End of else \*/

- printf("Exit <factor>\n");;

-} /\* End of function factor \*/

\ No newline at end of file

+parenthesis \*/

+else

+error();

+} /\* End of else \*/

+printf("Exit <factor>\n");;

+}

+/\* End of function factor \*/

\ No newline at end of file

diff --git a/front.in b/front.in

new file mode 100644

index 0000000..2fdfd51

--- /dev/null

+++ b/front.in

@@ -0,0 +1 @@

+a + b

\ No newline at end of file

commit 3cf8ec21eedf46eb7bb9d082868366b87c2ae763

Author: Utsab06 <utsab.khakurel@bison.howard.edu>

Date: Wed Mar 22 23:01:35 2017 -0400

front.c: Initial Lexical Analyzer system code and recursive-descent subprogram

diff --git a/a.out b/a.out

new file mode 100755

index 0000000..b343b18

Binary files /dev/null and b/a.out differ

diff --git a/front.c b/front.c

new file mode 100644

index 0000000..b0d07d9

--- /dev/null

+++ b/front.c

@@ -0,0 +1,252 @@

+/\* front.c - a lexical analyzer system for simple

+ arithmetic expressions \*/

+

+#include <stdio.h>

+#include <ctype.h>

+

+/\* Global declarations \*/

+/\* Variables \*/

+int charClass;

+char lexeme [100];

+char nextChar;

+int lexLen;

+int token;

+int nextToken;

+FILE \*in\_fp, \*fopen();

+

+/\* Function declarations \*/

+void addChar();

+void getChar();

+void getNonBlank();

+int lex();

+

+/\* Character classes \*/

+#define LETTER 0

+#define DIGIT 1

+#define UNKNOWN 99

+

+/\* Token codes \*/

+#define INT\_LIT 10

+#define IDENT 11

+#define ASSIGN\_OP 20

+#define ADD\_OP 21

+#define SUB\_OP 22

+#define MULT\_OP 23

+#define DIV\_OP 24

+#define LEFT\_PAREN 25

+#define RIGHT\_PAREN 26

+

+/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

+

+/\* main driver \*/

+main() {

+

+/\* Open the input data file and process its contents \*/

+ if ((in\_fp = fopen("front.in", "r")) == NULL)

+ printf("ERROR - cannot open front.in \n");

+ else {

+ getChar();

+ do {

+ lex();

+ } while (nextToken != EOF);

+ }

+}

+

+/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

+/\* lookup - a function to lookup operators and parentheses

+ and return the token \*/

+

+int lookup(char ch) {

+ switch (ch) {

+ case '(':

+ addChar();

+ nextToken = LEFT\_PAREN;

+ break;

+

+ case ')':

+ addChar();

+ nextToken = RIGHT\_PAREN;

+ break;

+

+ case '+':

+ addChar();

+ nextToken = ADD\_OP;

+ break;

+

+ case '-':

+ addChar();

+ nextToken = SUB\_OP;

+ break;

+

+ case '\*':

+ addChar();

+ nextToken = MULT\_OP;

+ break;

+

+ case '/':

+ addChar();

+ nextToken = DIV\_OP;

+ break;

+

+ default:

+ addChar();

+ nextToken = EOF;

+ break;

+ }

+ return nextToken;

+}

+

+/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

+/\* addChar - a function to add nextChar to lexeme \*/

+

+void addChar() {

+ if (lexLen <= 98) {

+ lexeme[lexLen++] = nextChar;

+ lexeme[lexLen] = 0;

+ }

+ else

+ printf("Error - lexeme is too long \n");

+}

+

+/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

+/\* getChar - a function to get the next character of

+ input and determine its character class \*/

+

+void getChar() {

+ if ((nextChar = getc(in\_fp)) != EOF) {

+ if (isalpha(nextChar))

+ charClass = LETTER;

+ else if (isdigit(nextChar))

+ charClass = DIGIT;

+ else charClass = UNKNOWN;

+ }

+ else

+ charClass = EOF;

+}

+

+/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

+/\* getNonBlank - a function to call getChar until it

+ returns a non-whitespace character \*/

+

+void getNonBlank() {

+ while (isspace(nextChar))

+ getChar();

+}

+

+

+/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

+/\* lex - a simple lexical analyzer for arithmetic

+ expressions \*/

+

+int lex() {

+ lexLen = 0;

+ getNonBlank();

+ switch (charClass) {

+

+ /\* Parse identifiers \*/

+ case LETTER:

+ addChar();

+ getChar();

+ while (charClass == LETTER || charClass == DIGIT) {

+ addChar();

+ getChar();

+ }

+ nextToken = IDENT;

+ break;

+

+/\* Parse integer literals \*/

+ case DIGIT:

+ addChar();

+ getChar();

+ while (charClass == DIGIT) {

+ addChar();

+ getChar();

+ }

+ nextToken = INT\_LIT;

+ break;

+

+/\* Parentheses and operators \*/

+ case UNKNOWN:

+ lookup(nextChar);

+ getChar();

+ break;

+

+/\* EOF \*/

+ case EOF:

+ nextToken = EOF;

+ lexeme[0] = 'E';

+ lexeme[1] = 'O';

+ lexeme[2] = 'F';

+ lexeme[3] = 0;

+ break;

+

+ } /\* End of switch \*/

+ printf("Next token is: %d, Next lexeme is %s\n",

+ nextToken, lexeme);

+ return nextToken;

+} /\* End of function lex \*/

+

+ /\* expr

+ Parses strings in the language generated by the rule:

+ <expr> -> <term> {(+ | -) <term>} \*/

+

+void expr() {

+ printf("Enter <expr>\n");

+/\* Parse the first term \*/

+ term();

+/\* As long as the next token is + or -, get

+ the next token and parse the next term \*/

+ while (nextToken == ADD\_OP || nextToken == SUB\_OP) {

+ lex();

+ term();

+ }

+ printf("Exit <expr>\n");

+} /\* End of function expr \*/

+

+ /\* term

+ Parses strings in the language generated by the rule:

+ <term> -> <factor> {(\* | /) <factor>)

+ \*/

+void term() {

+ printf("Enter <term>\n");

+/\* Parse the first factor \*/

+ factor();

+/\* As long as the next token is \* or /, get the

+ next token and parse the next factor \*/

+ while (nextToken == MULT\_OP || nextToken == DIV\_OP) {

+ lex();

+ factor();

+ }

+ printf("Exit <term>\n");

+} /\* End of function term \*/

+

+ /\* factor

+ Parses strings in the language generated by the rule:

+ <factor> -> id | int\_constant | ( <expr )

+ \*/

+void factor() {

+ printf("Enter <factor>\n");

+/\* Determine which RHS \*/

+ if (nextToken == IDENT || nextToken == INT\_LIT)

+ /\* Get the next token \*/

+ lex();

+/\* If the RHS is ( <expr>), call lex to pass over the

+ left parenthesis, call expr, and check for the right

+ parenthesis \*/

+ else {

+ if (nextToken == LEFT\_PAREN) {

+ lex();

+ expr();

+ if (nextToken == RIGHT\_PAREN)

+ lex();

+ else

+ error();

+ } /\* End of if (nextToken == ... \*/

+

+/\* It was not an id, an integer literal, or a left

+ parenthesis \*/

+ else

+ error();

+ } /\* End of else \*/

+ printf("Exit <factor>\n");;

+} /\* End of function factor \*/

\ No newline at end of file

Instructions to compile: gcc -o parser front.c

./parser front.in