**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**PROGRAMMING LANGUAGE**

**2020-2021 SPRING SEMESTER**

**PROJECT-1 REPORT**

Writing a Lexical Analyzer for The BigAdd Language

**Delivery Date: 08/06/2021**

**PREPARED BY**

Esra Duman 05190000076

Selin Paksoy 05190000029

Table Of Contents

[Analysis 3](#_Toc73726228)

[Design 3](#_Toc73726229)

[Source Code 4](#_Toc73726230)

[Screenshots 26](#_Toc73726231)

[Catalog for Programmers 27](#_Toc73726232)

# Analysis

This project requests a lexical analyzer for BigAdd Language. The lexical analyzer must take input from a user to use the filename. The input file will read from the code and the lexical analyzer will analyze its demands. For understanding “BigAdd Language”, the code must have known what this language looks like. The lexical analyzer should recognize the keywords, identifiers, constants, end of line symbols, etc. If the lexical analyzer encounters an error, it should print an error message on the screen. Also, the lexical analyzer must write what reads on the file to a new file that has the same name but a different extension.

BigAdd language is a language that consists only of int-type data. This language includes assignment, addition, subtraction, loops, code blocks and comments. In addition, there are string constants written within the characters “ “ in the language.

# Design

We tried to design this project with the most efficient way. Firstly, we take the name of the file that the user wants to read as input. The code opens the file by adding an extension to this input value. Then, it writes what it reads from the BigAdd language code to a file with the same file name but a different extension. To analyze the code in the file, the text read is split into lines. Then, each line is checked with the "containPar" method to see if it contains a comment line. If there is a comment line, it is deleted. After this process, if there is an end of line symbol in the code, this symbol is also deleted from the line with the “strEndsWith” method. If there is not an end of line symbol, the error output is printed on the screen.

The remaining line of code is sent to the "analyze" method for analysis. In the "analyze" method, lines are split into words. Then each split word is classified, the code is analyzed. We have created methods such as "isKeyword", "isInt", "isIntVariable", "isVariable" to classify words. In these methods, we made the definitions in accordance with the BigAdd language.

If there is an error or deficiency in the code read, this error is printed on the screen as output.

Part of the code does not work because previously defined integer variables in the "analyze" method cannot be found in the "variables" list. "loop" and "move" parts do not work because we could not solve this problem.

Also, we could not write the analysis results into the "newFile" that file we created. We tried to use fprintf, fputs and fputc methods, but they did not work. We realized that these methods worked when we changed the file extension to "txt". However, we did not delete the lines written into the file so that there is no deficiency in the code. We could not write the results to the file, so we printed them as output to the screen.

# Source Code

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <assert.h>

#include <ctype.h>

// Esra Duman 05190000076

// Selin Paksoy 05190000029

char\* variables[1000] = {}; //our list of variables

int intList[1000] = {0}; //our list of integers

int countVar = 0, countInt = 0, countEndOfLine = 0, countParanthesis = 0; //counters for output

char commentLine[100]; char a[100]; char b[100];

int strEndsWith(const char \*str) //checks if a string ends with "."

{

const char compare[] = ".";

size\_t sLen = strlen(str);

size\_t cLen = strlen(compare);

if (cLen > sLen) return 1;

return strcmp(str + sLen - cLen, compare); //it returns 0 for match

}

char\* removeCom(char \*code, int state){

char newCode [1000] = { };

char \*o, \*c; //open and close parenthesis

int indexFirst, indexSec, codeLen ,comLen,tempValue;

codeLen = strlen(code);

int i;

if(state==2){

o = strchr(code, '{');

c = strchr(code, '}');

indexFirst = (int)(o - code);

indexSec = (int)(c -code);

comLen = indexSec - indexFirst;

for(i = 0; i < codeLen; i++){ //if parenthesis starts and ends in the same line the code will delete comment part

tempValue = i - comLen;

if(i<indexFirst)

newCode[i] = code[i];

else if (i>indexSec)

newCode[tempValue] = code[i];

else

continue;

}

}

else if(state == 1){ // if parenthesis starts in this line, the code will take before the opening symbol

o = strchr(code, '{');

indexFirst = (int)(o - code);

for(i = 0; i < indexFirst; i++){

newCode[i] = code[i];

}

}

else if(state == 0){ //if parenthesis ends in this line, the code will take after the closing symbol

c = strchr(code, '}');

if (c>=0){

int x = 0;

for(i = c; i < codeLen; i++){

newCode[x] = code[i];

x++;

}

}

}

return newCode;

}

bool isKeyword(char\* newStr) { //checks if it is a keyword or not

//our special keywords

if (!strcmp(newStr, "int") || !strcmp(newStr, "move") || !strcmp(newStr, "to") || !strcmp(newStr, "loop") || !strcmp(newStr, "times") ||

!strcmp(newStr, "out") || !strcmp(newStr, "add") || !strcmp(newStr, "sub") || !strcmp(newStr, "from") || !strcmp(newStr, "newline"))

return true;

else

return false;

}

bool isIntVariable(char\* str) { //adds the identifiers to our list of variables

int i;

for (i = 0; i < countVar; i++) {

if (!strcmp(str, variables[i]))

return true;

}

return false;

}

bool isVariable(char\* newStr){ //checks to correct name for identifiers

int strLen = strlen(newStr);

int i;

if (strLen > 20) //the name of identifier must be shorter than 20

return false;

for (i = 0; i < strLen; i++) {

if (!((newStr[i] >= 'A' && newStr[i] <= 'Z') || (newStr[i] >= 'a' && newStr[i] <= 'z') || (newStr[i] == '\_')))

return false;

if (newStr[0] == '\_')

return false;

}

return true;

}

bool isInt(char\* str) { //checks to the intConstant

bool state = true;

int i;

if (strlen(str) > 100) //if the length of str is bigger than 100, it's not an integer

state = false;

for (i = 0; i < strlen(str); i++) {

if (i > 0) {

if (str[i] == '-' || strlen(str) == 1) //integer cant contain '-' symbol

state = false;

}

else if (!(isdigit(str[i]) || str[i] == '-')) // checks for negativity and other symbols

state = false;

}

return state;

}

int containPar(char \*word, int parState, int countLine){ // checks if the line contains parenthesis

if(strstr(word, "{")!= NULL && strstr(word, "}") !=NULL){

char \*o, \*c; //open and close parenthesis

int indexFirst, indexSec;

o = strchr(word, '{');

c = strchr(word, '}');

indexFirst = (int)(o - word); //start index

indexSec = (int)(c -word); //end index

if(indexFirst < indexSec){

parState = 2;

}

else if(parState == 1){ //indexFirst > indexSec

parState = 1; //open parenthesis closed new parenthesis opened

}

else if(parState == 0 ){

printf("Error on line %d! Parenthesis was not open!", countLine);

parState =1; //parenthesis is open right now

}

}

else if (parState ==0 && strstr(word, "{")!= NULL) //parenthesis opened

parState = 1;

else if (parState ==1 && strstr(word, "}") !=NULL)

parState = 0; //parenthesis closed

else if ((parState==0 || parState == 2) && strstr(word, "}") != NULL){

printf("Error on line %d! Parenthesis was not open!", countLine);

parState = 0;

}

else if (parState == 1 && strstr(word, "{"))

printf("Error on line %d! Parenthesis already opened!", countLine);

return parState;

}

void analyze(char\* line, FILE \*writeFile, int countLine) { //method for parsing lines and analyzing the words

int wordCount = 0;

char \*word[1000];

char \*splitLine = strtok(line, " ");

int endOf;

while (splitLine != NULL){ //splitting lines to words

if(strcmp(splitLine, "\0")) {

word[wordCount++] = splitLine;

splitLine = strtok(NULL, " ");

}

}

char finalWord[1000];

strcpy(finalWord, word[wordCount-1]);

//checks the end of line symbol

endOf = strEndsWith(finalWord); //if it equals to zero, it matched.

if (endOf==0)

word[wordCount-1][strlen(word[wordCount-1])-1] = '\0';

int y=0;

for (y = 0; y < wordCount; y++){

//keyword int

if (!strcmp(word[y], "int")) {

printf("Keyword int\n");

fprintf(writeFile, "Keyword int\n");

char \*nextWord = word[y+1];

if (isVariable(nextWord)){

countInt++;

countVar++;

intList[countInt] = 0; //adding the identifier to our lists

variables[countVar] = word[y+1];

printf("Identifier %s \n", nextWord);

fprintf(writeFile, "Identifier \n");

}

else if (isKeyword(nextWord))

printf("Error on line %d! %s is a keyword. Identifier's name must be different from the keywords.\n", countLine, nextWord);

else if (nextWord == NULL)

printf("Error on line %d!. Identifier of the integer is expected.\n", countLine);

else

printf("Error on line %d! %s is not an identifier! Identifier is expected.", countLine,nextWord);

}

//keyword move

else if (!strcmp(word[y], "move")) {

printf("Keyword move \n");

fprintf(writeFile, "Keyword move \n");

//intConstant or identifier

int moveInt = 0;

if (word[y+1] == NULL)

printf("Error on line %d! Integer or variable are expected.\n", countLine);

else if (isKeyword(word[y+1]))

printf("Error on line %d! %s is a keyword. Integer or variable are expected.\n", countLine, word[y+1]);

else if (isInt(word[y+1])) {

printf("IntConstant %s\n", word[y+1]);

moveInt = atoi(word[y+1]);

}

else if (isIntVariable(word[y+1])) {

printf("Identifier %s\n", word[y+1]);

fprintf(writeFile, "Identifier %s\n", word[y+1]);

int ik;

for ( ik = 0; ik < countVar; ik++) {

if (!strcmp(variables[ik], word[y+1])) {

moveInt = intList[ik];

break;

}

}

}

else

printf("Error on line %d! %s is not a variable or an integer. Integer or variable are expected.\n", countLine, word[y+1]);

//keyword to

if (!strcmp(word[y+2], "to")){

printf("Keyword to \n");

fprintf(writeFile, "Keyword to \n");

}

else

printf("Error on line %d! Keyword 'to' is expected.\n", countLine);

//identifier

if (isKeyword(word[y+3]))

printf("Error on line %d! %s is a keyword. Identifier's name must be different from the keywords.\n", countLine, word[y+3]);

else if (isIntVariable(word[y+3])) {

printf("Identifier %s \n", word[y+3]);

fprintf(writeFile, "Identifier %s \n", word[y+3]);

int ij;

for (ij = 0; ij < countVar; ij++) {

if (!strcmp(variables[ij], word[y+3])) {

intList[ij] = moveInt;

break;

}

}

}

else

printf("Error on line %d! %s is not an identifier. The identifier is expected.\n", countLine, word[y+3]);

}

//keyword add

else if (!strcmp(word[y], "add")) {

printf("Keyword add \n");

fprintf(writeFile, "Keyword add \n");

//intConstant or identifier

int addInt = 0;

if (word[y+1] == NULL)

printf("Error on line %d! Integer or variable are expected. \n", countLine);

else if (isKeyword(word[y+1]))

printf("Error on line %d! %s is a keyword! Integer or variable are expected.\n", countLine, word[y+1]);

else if (isInt(word[y+1])) {

printf("'IntConstant %s \n", word[y+1]);

fprintf(writeFile, "'IntConstant %s \n", word[y+1]);

addInt = atoi(word[y+1]);

}

else if (isIntVariable(word[y+1])) {

printf("Identifier %s \n", word[y+1]);

fprintf(writeFile, "Identifier %s \n", word[y+1]);

int il;

for (il = 0; il < countVar; il++) {

if (!strcmp(variables[il], word[y+1])) {

addInt = intList[il];

break;

}

}

}

else

printf("Error on line %d! %s is not a variable or an integer. Integer or variable are expected.\n", countLine, word[y+1]);

//keyword to

if (!strcmp(word[y+2], "to")) {

printf("Keyword to \n");

fprintf(writeFile, "Keyword to \n");

}

else

printf("Error on line %d! Keyword %s is expected.\n", countLine, word[y+2]);

//intConstant

if (word[y+3] == NULL)

printf("Error on line %d! Variable is expected.\n", countLine);

else if (isKeyword(word[y+3]))

printf("Error on line %d! %s is a keyword. Variable is expected.\n", countLine, word[y+3]);

else if (isIntVariable(word[y+3])) {

printf("IntConstant %s \n", word[y+3]);

fprintf(writeFile, "IntConstant %s \n", word[y+3]);

int iz;

for (iz = 0; iz < countVar; iz++) {

if (!strcmp(variables[iz], word[y+3])) {

intList[iz] += addInt;

break;

}

}

}

else

printf("Error on line %d! %s is not a variable. Variable is expected.\n", countLine, word[y+3]);

}

//keyword sub

else if (!strcmp(word[y], "sub")) {

printf("Keyword sub \n");

fprintf(writeFile, "Keyword sub \n");

//intConstant, identifier

int subInt = 0;

if (word[y+1] == NULL)

printf("Error on line %d! Integer or variable are expected.\n", countLine);

else if (isKeyword(word[y+1]))

printf("Error on line %d! %s is a keyword. Integer or variable are expected.\n", countLine, word[y+1]);

else if (isInt(word[y+1])) {

printf("IntConstant %s\n", word[y+1]);

fprintf(writeFile, "IntConstant %s\n", word[y+1]);

subInt = atoi(word[y+1]);

}

else if (isIntVariable(word[y+1])) {

printf("Identifier %s\n", word[y+1]);

fprintf(writeFile, "Identifier %s\n", word[y+1]);

int i;

for ( i = 0; i < countVar; i++) {

if (!strcmp(variables[i], word[y+1])) {

subInt = intList[i];

break;

}

}

}

else

printf("Error on line %d! %s is not a variable or an integer. Integer or variable are expected.\n", countLine, word[y+1]);

//keyword to

if (!strcmp(word[y+2], "from")){

printf("Keyword from \n");

fprintf(writeFile, "Keyword from \n");

}

else

printf("Error on line %d! Keyword 'from' is expected.\n", countLine);

//identifier

if (word[y+3] == NULL)

printf("Error on line %d! The identifier is expected.\n", countLine);

else if (isKeyword(word[y+3]))

printf("Error on line %d! %s is a keyword. The identifier is expected.\n", countLine, word[y+3]);

else if (isIntVariable(word[y+3])) {

printf("Identifier %s", word[y+3]);

fprintf(writeFile, "Identifier %s", word[y+3]);

int is;

for (is = 0; is < countVar; is++) {

if (!strcmp(variables[is], word[y+3])) {

intList[is] -= subInt;

break;

}

}

}

else

printf("Error on line %d! %s is not a variable. The identifier is expected.\n", countLine, word[y+3]);

}

//keyword newline

else if (!strcmp(word[y], "newline")){

printf("Keyword newline \n");

fprintf(writeFile, "Keyword newline \n");

}

//keyword out

else if (!strcmp(word[y], "out")) {

printf("Keyword out \n");

fprintf(writeFile, "Keyword out \n");

//intConstant, identifier, stringConstant

if (word[y+1] == NULL)

printf("Error on line %d! IntConstant, identifier or StringConstant are expected.\n", countLine);

else if (isInt(word[y+1])){

printf("IntConstant %s\n", word[y+1]);

fprintf(writeFile,"IntConstant %s\n", word[y+1]);

}

else if (isIntVariable(word[y+1])){

printf("Identifier %s\n", word[y+1]);

fprintf(writeFile, "Identifier %s\n", word[y+1]);

}

else{

printf("StringConstant %s\n", word[y+1]);

fprintf(writeFile, "StringConstant %s\n", word[y+1]);

}

}

//keyword loop

else if (!strcmp(word[y], "loop")) {

printf("Keyword loop \n");

//intConstant or identifier

int loopInt = 0;

if (word[y+1] == NULL)

printf("Error on line %d! Integer or variable are expected.\n", countLine);

else if (isKeyword(word[y+1]))

printf("Error on line %d! %s is a keyword. Integer or variable are expected.\n", countLine, word[y+1]);

else if (isInt(word[y+1])) {

printf("IntConstant %s \n", word[y+1]);

loopInt = atoi(word[y+1]);

}

else if (isIntVariable(word[y+1])) {

printf("Identifier %s \n", word[y+1]);

fprintf(writeFile, "Identifier %s \n", word[y+1]);

int it;

for (it = 0; it < countVar; it++) {

if (!strcmp(variables[it], word[y+1])) {

loopInt = intList[it];

break;

}

}

}

else

printf("Error on line %d! %s is not an integer or a variable. An integer or a variable are expected.\n", countLine, word[y+1]);

//keyword times

if (!strcmp(word[y+2], "times")){

printf("Keyword times \n");

fprintf(writeFile, "Keyword times \n");

}

else

printf("Keyword 'times' is expected.\n");

//code part

int iy;

for (iy = 0; iy < loopInt; iy++) {

char newCode[100] = "";

int iyj;

for (iyj = 3; iyj < wordCount; iyj++) {

strcat(newCode, " ");

strcat(newCode, word[iyj]);

}

strcat(newCode, ".");

analyze(newCode, writeFile, countLine);

}

}

else if (endOf == 0 ){ //the end of line symbol

printf("EndOfLine \n");

fprintf(writeFile, "EndOfLine \n");

}

else

printf("End of line is expected.\n");

}

}

int subLine(char \*source, int from, int n, char \*target) {

int length, i;

for (length = 0; source[length] != '\0'; length++);

if (from > length) {

printf("Invalid index. \n");

return 1;

}

if ((from + n) > length) {

n = (length - from);

}

for (i = 0; i < n; i++) {

target[i] = source[from + i];

}

target[i] = '\0';

return 0;

}

void isComment(char\* str) { //checks if it contains a comment

bool trueComment = false;

int ij;

int i;

for (i = 0; i < strlen(str); i++) {

if (str[i] == '{') {

for (ij = i; ij < strlen(str); ij++) {

if (str[ij] == '}') {

subLine(str, i + 1, ij - i - 1, commentLine);

printf("Comment Line: '%s'\n", commentLine);

subLine(str, 0, i, a);

subLine(str, ij + 1, strlen(str), b);

strcat(a, b);

strcpy(str, a);

trueComment = true;

isComment(str);

}

}

if (!trueComment) {

printf("Comment Line is not completed.\n");

exit(0);

}

}

}

}

int main() {

FILE \*fp;

FILE \*newFile;

long lSize;

char \*code;

char fileName[20];

char fileName2[20];

int k = 0;

printf("Enter the name of file: "); //asks the user for the name of the file

scanf("%s",fileName); //keeps the name of file

strcpy(fileName2, fileName); //copy the name to create new file "fileName2"

while(fileName[k]!='\0')

k++;

//adding the extension

fileName[k]='.'; //for reading to file

fileName[k+1]='b';

fileName[k+2]='a';

fileName[k+3]='\0';

fileName2[k] = '.'; //for writing to file

fileName2[k+1] = 'l';

fileName2[k+2] = 'x';

fileName2[k+3] = '\0';

fp=fopen(fileName,"r");

if (!fp) perror(fileName), exit(1);

else printf("\nThe given file '%s' is found. Reading...\n", fileName);

newFile = fopen(fileName2, "w");

if (!newFile) perror(fileName2), exit(1);

else printf("\nA new file named '%s' has been created. Processing...\n", fileName2);

fseek(fp, 0L, SEEK\_END);

lSize = ftell(fp);

rewind(fp);

code = calloc(1, lSize + 1);

if (!code)

fclose(fp), fputs("memory fails", stderr), exit(1);

if (1 != fread(code, lSize, 1, fp))

fclose(fp), free(code), fputs("read fails", stderr), exit(1);

int lineCount = 0;

char \*line[1000];

char \*splitCode = strtok(code, "\n");

while (splitCode != NULL) {

line[lineCount++] = splitCode;

splitCode = strtok(NULL, "\n");

}

printf("\nLine Count: %d \n\n", lineCount);

int x;

int state = 0; //the parenthesis of comment state

isComment(code);

for (x = 0; x < lineCount; x++){

state = containPar(line[x], state, lineCount); //checks if it contains a parenthesis

if(state!=0)

strcpy(line[x],removeCom(line[x], state)); //removes comment into the line

analyze(line[x], newFile, lineCount);

}

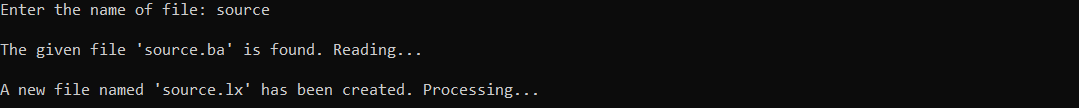
fclose(fp);

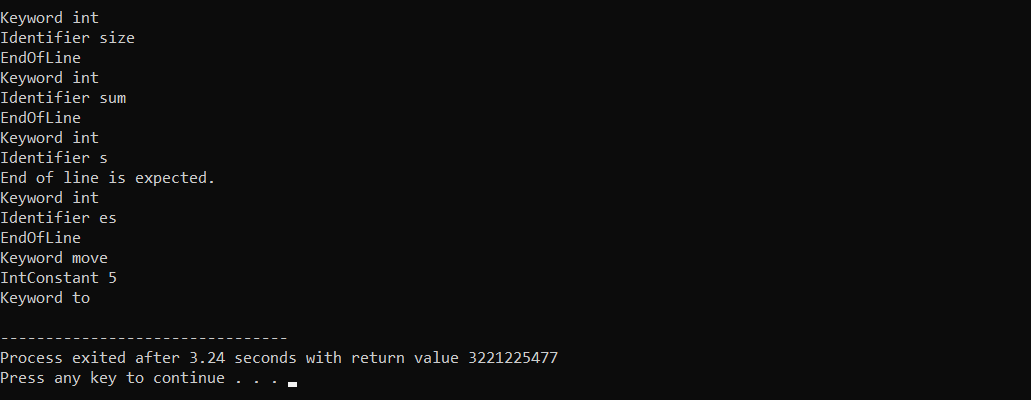
fclose(newFile);

}

# Screenshots







# Catalog for Programmers

Analysis: 2 Days

Design: 4 Days

Coding and Decoding: 5 Days

Testing the code and preparing a report: 5 Hours