

Ege University

Department of Computer Engineering

Programming Language 2020-2021 Spring Semester Project

**Project 1: Writing a lexical analyzer for the BigAdd Language**

Name: Sami Tuğal ID: 05180000086

Name: Ahmet Anıl Özsoy ID: 05180000054

İçidekiler Tablosu

**Data Types3**

**Variables4**

**Assigment Statement5**

**Addition Statement6**

**Subtraction Statement7**

**Output Statement 8**

**Loop9**

**CodeBlock12**

**Comments12**

**Code12**

**Results37**

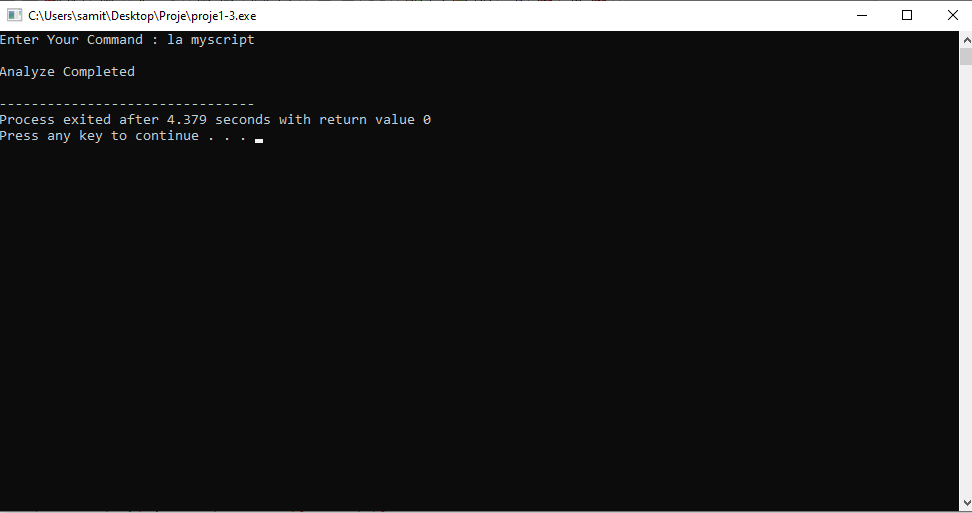
**Project 1: Writing a lexical analyzer for the BigAdd Language**

Last version of DevC++, Language: C

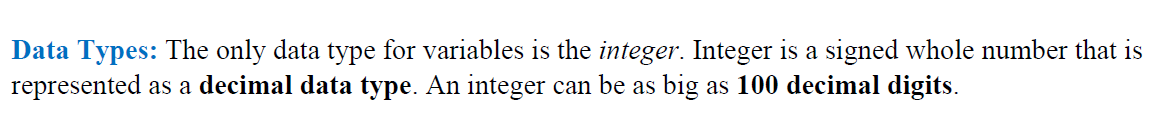
How It Works:

**Highly recommended** : Putting spaces before commas.

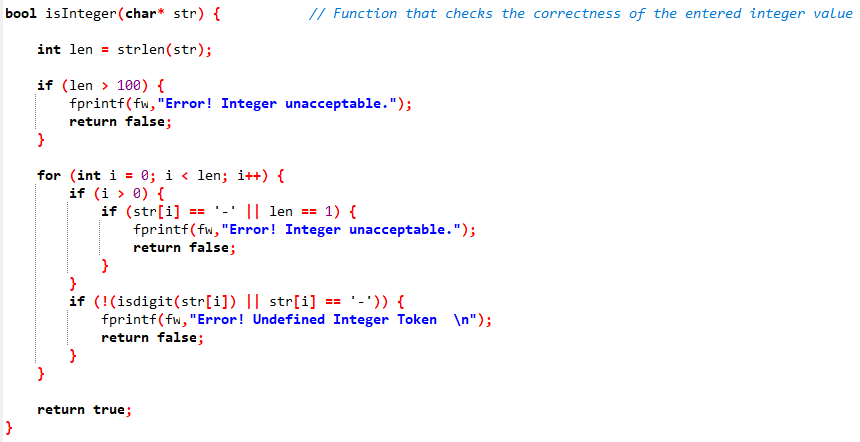
You must enter the correct Launch code



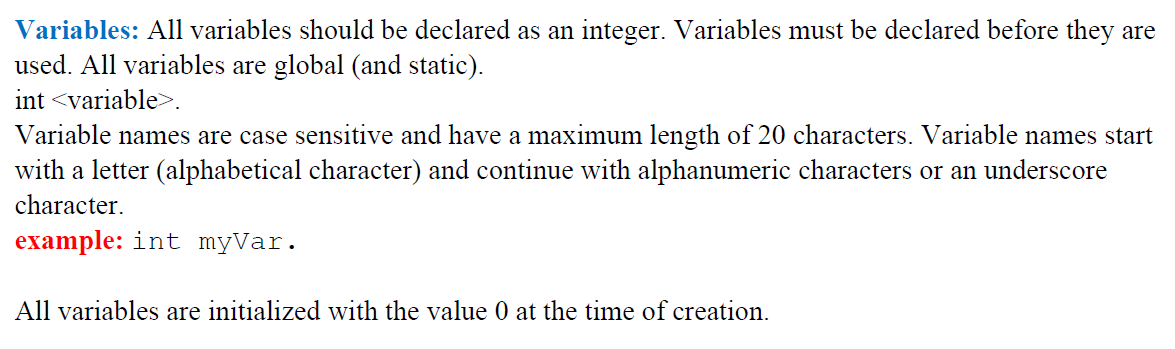
## **1.a** **Data Types**



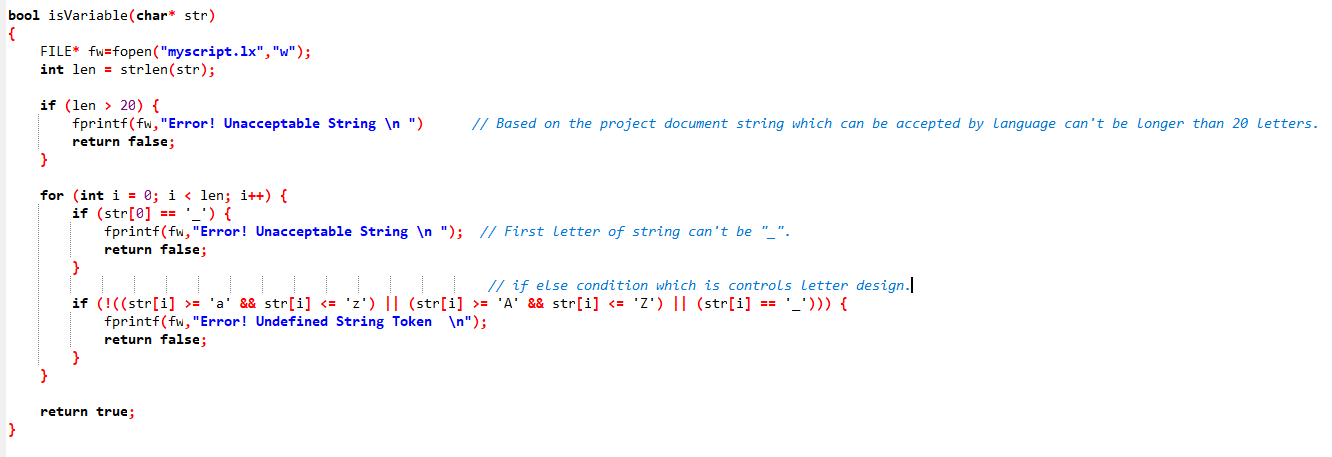
Function that checks the correctness of the entered integer value



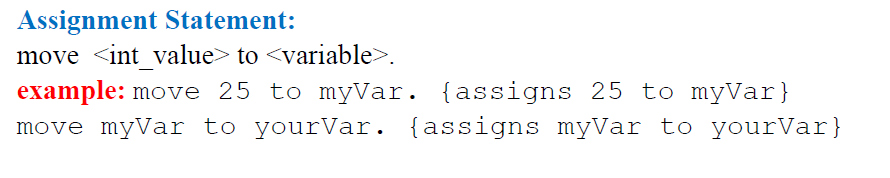
## 1.b Variables}

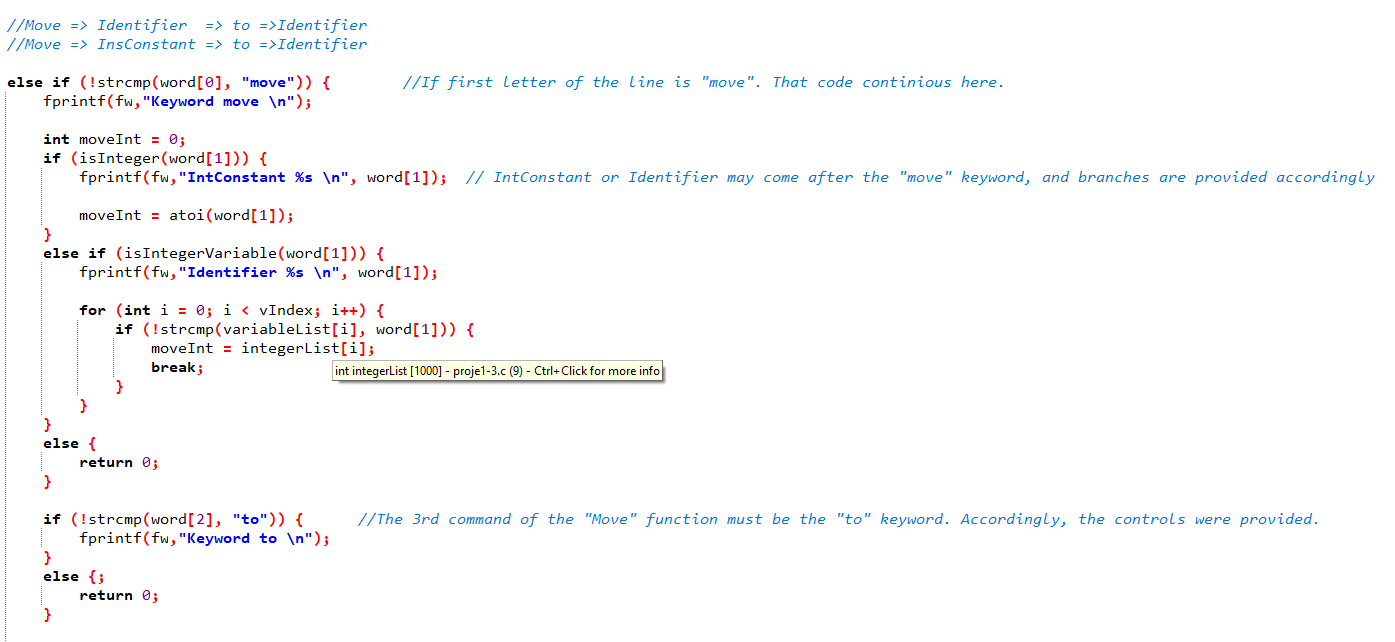


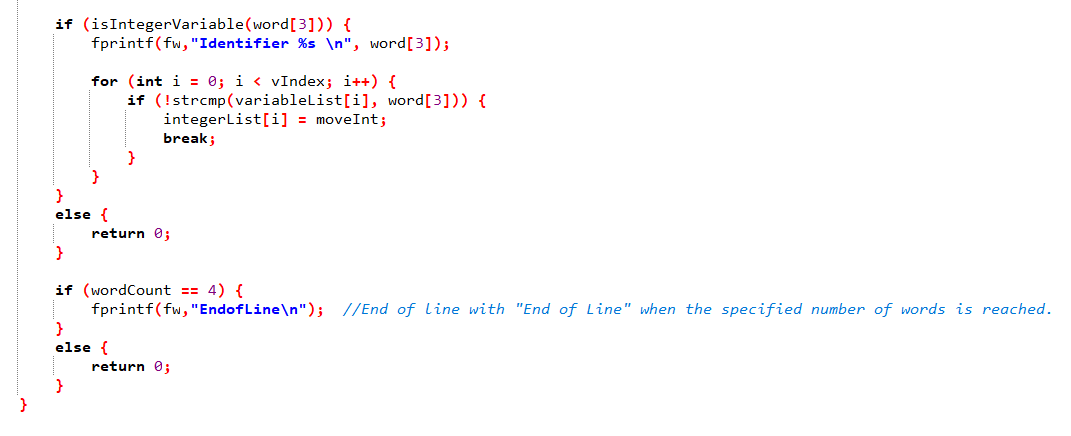
Function that checks the correctness of the entered string value .



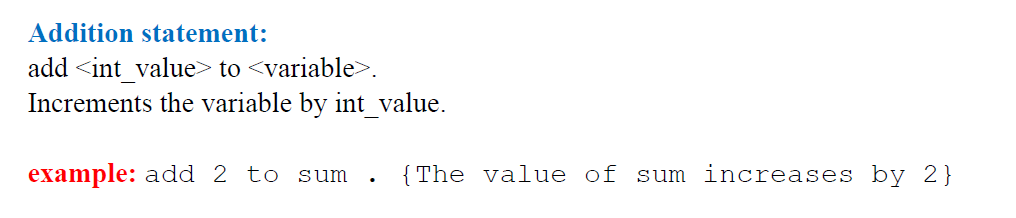
## 1.c Assigment Statement:



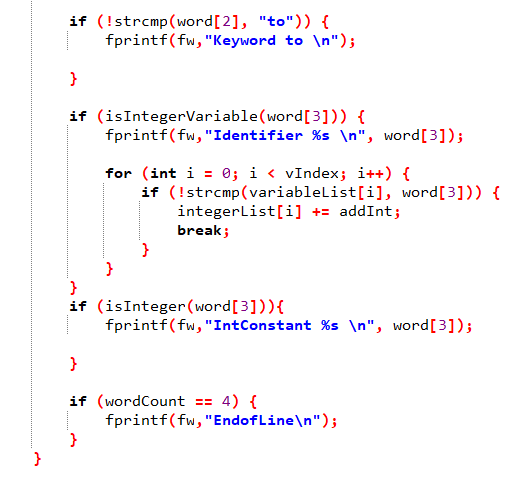




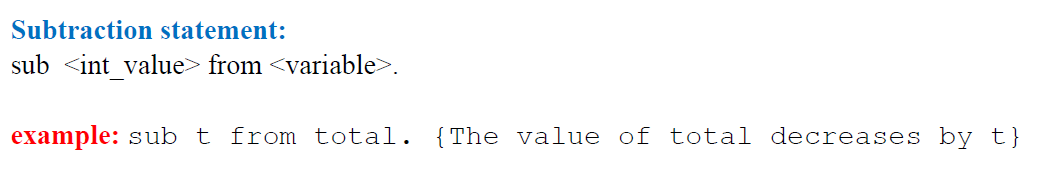
### 1.d Addition Statement :



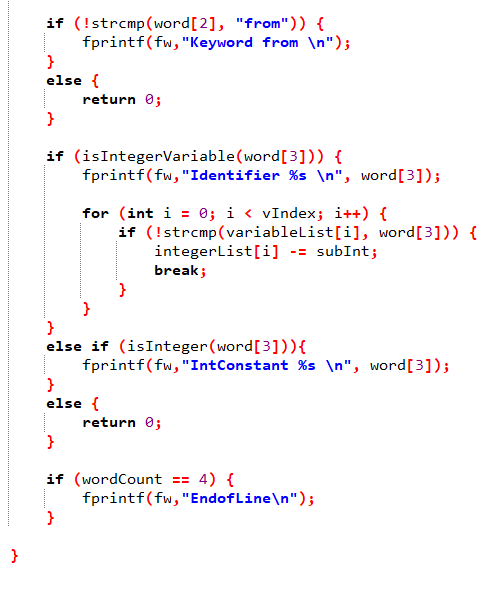




## 1.e Subtraction Statement

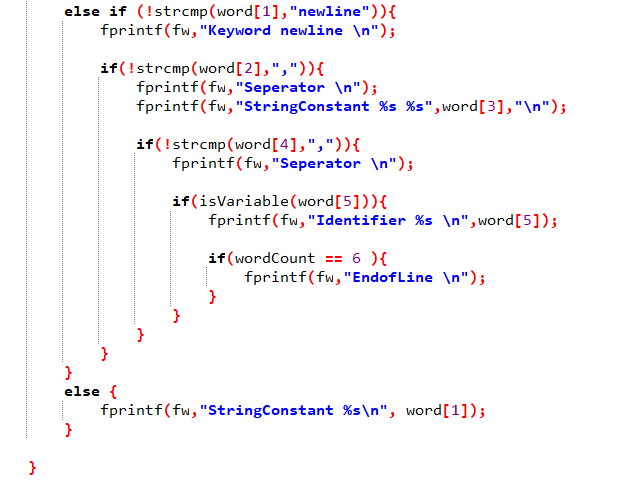
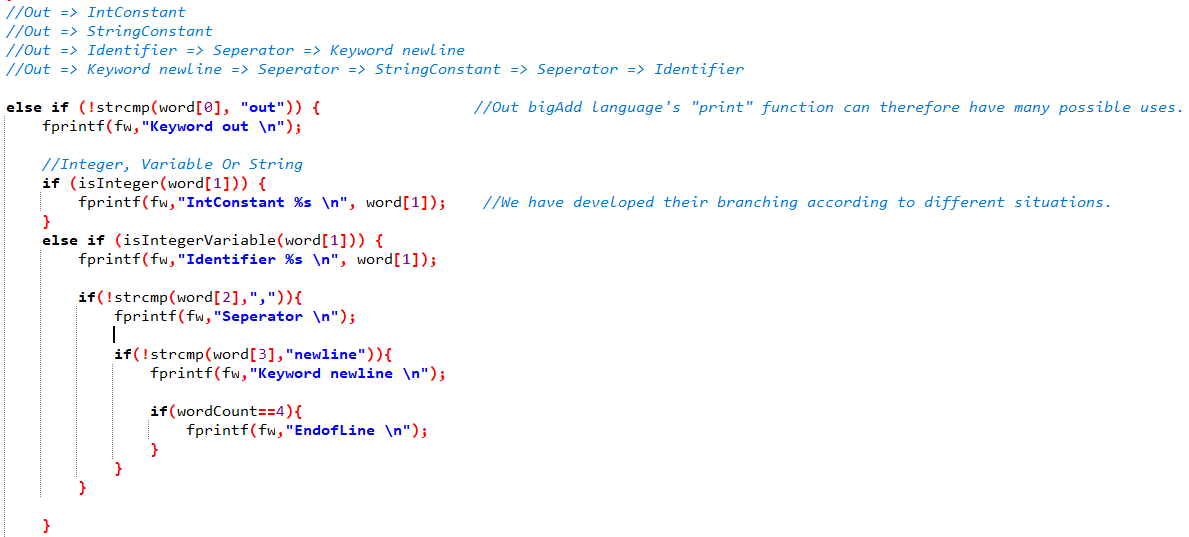




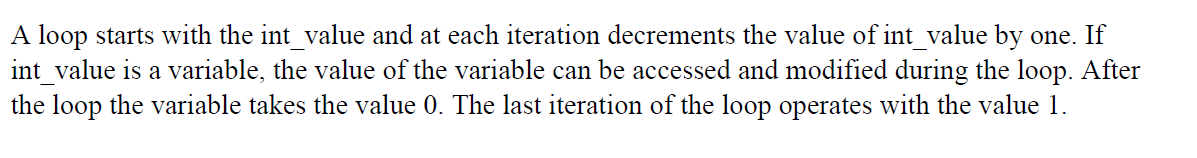


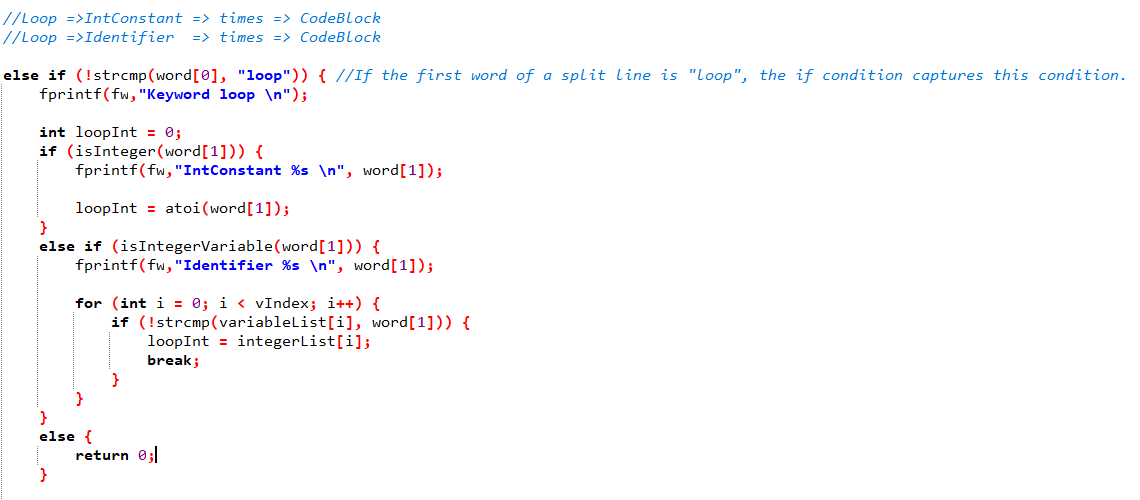
## 1.f Output Statement

### 

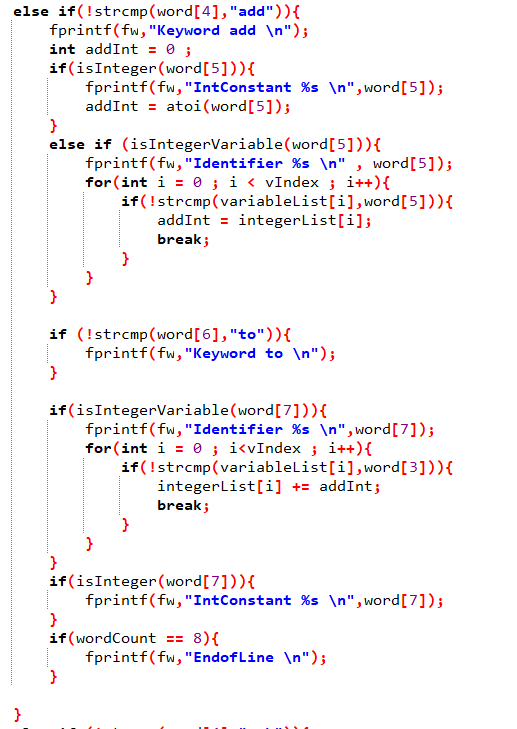


## 1.g **Loop**

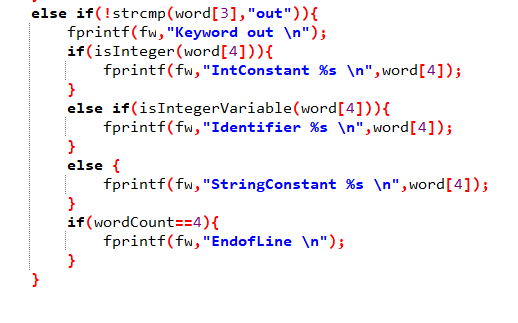




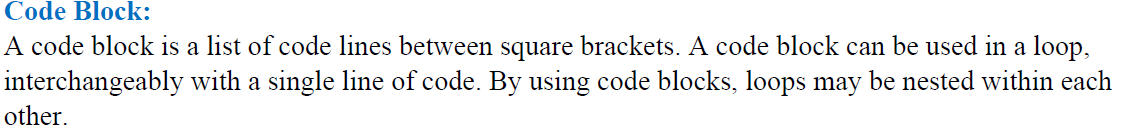


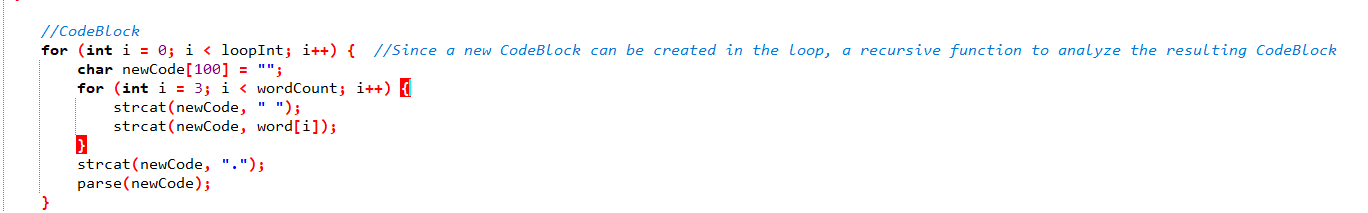




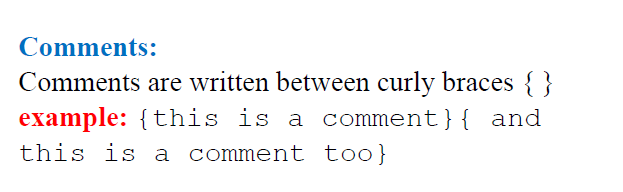


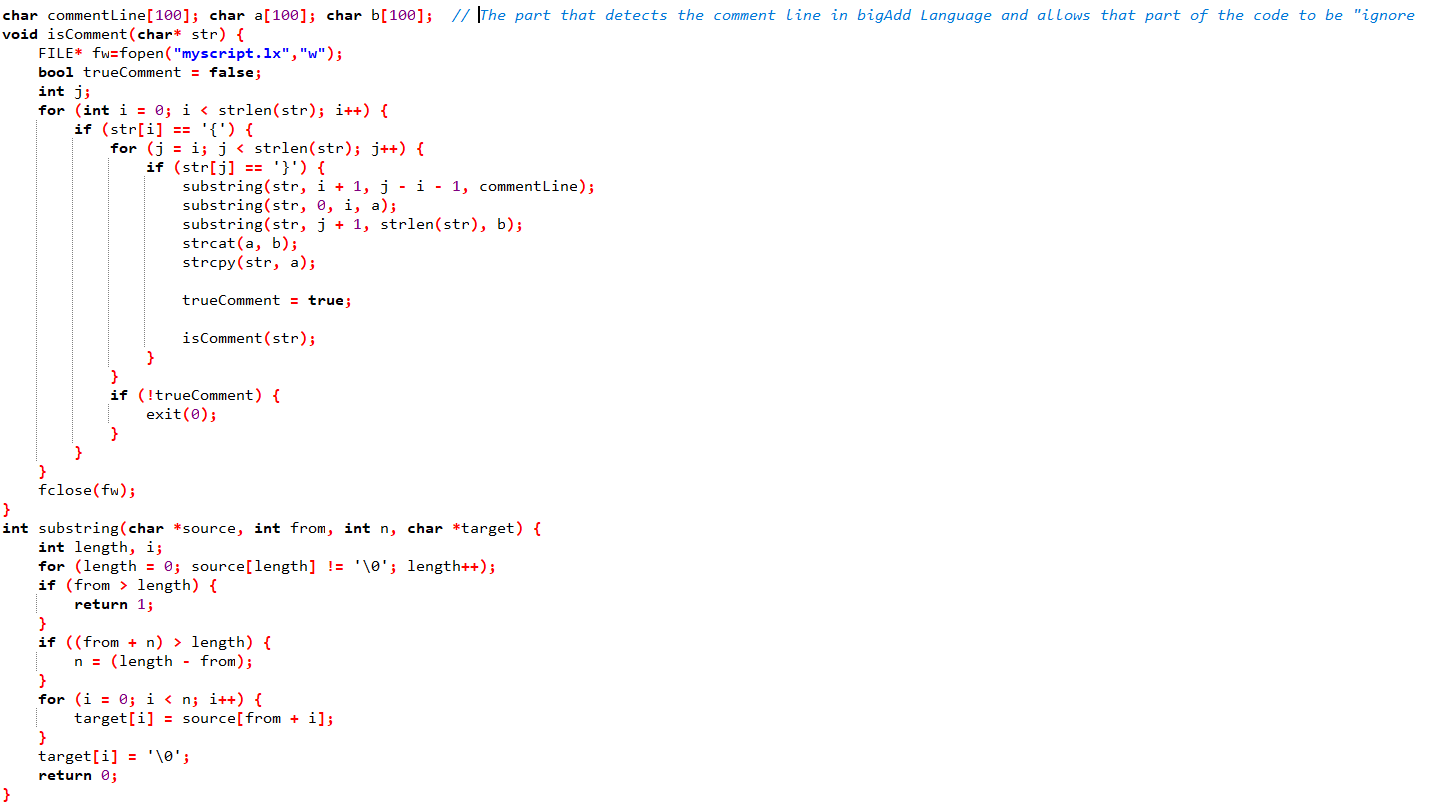
## 1.h **CodeBlock**





## 1.I **Comments**





## 2.A **Code**

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <assert.h>

#include <string.h>

char \*variableList[1000]; int vIndex = 0;

int integerList[1000] = { 0 }; int iIndex = 0;

bool isKeyword(char\* str) {

if (!strcmp(str, "int") ||

!strcmp(str, "move") || !strcmp(str, "to") || !strcmp(str, "add") ||

!strcmp(str, "sub") || !strcmp(str, "from") ||

!strcmp(str, "loop") || !strcmp(str, "times") ||

!strcmp(str, "out") || !strcmp(str,"newline")) {

return true;

}

return false;

}

bool isInteger(char\* str) { // Function that checks the correctness of the entered integer value

FILE\* fw=fopen("myscript.lx","w");

int len = strlen(str);

if (len > 100) {

fprintf(fw,"Error! Integer unacceptable.");

return false;

}

for (int i = 0; i < len; i++) {

if (i > 0) {

if (str[i] == '-' || len == 1) {

fprintf(fw,"Error! Integer unacceptable.");

return false;

}

}

if (!(isdigit(str[i]) || str[i] == '-')) {

fprintf(fw,"Error! Undefined Integer Token \n");

return false;

}

}

return true;

}

bool isIntegerVariable(char\* str) {

for (int i = 0; i < vIndex; i++) {

if (!strcmp(str, variableList[i])) {

return true;

}

}

return false;

}

bool isVariable(char\* str)

{

FILE\* fw=fopen("myscript.lx","w");

int len = strlen(str);

if (len > 20) {

fprintf(fw,"Error! Unacceptable String \n "); // Based on the project document string which can be accepted by language can't be longer than 20 letters.

return false;

}

for (int i = 0; i < len; i++) {

if (str[0] == '\_') {

fprintf(fw,"Error! Unacceptable String \n "); // First letter of string can't be "\_".

return false;

}

// if else condition which is controls letter design.

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

fprintf(fw,"Error! Undefined String Token \n");

return false;

}

}

return true;

}

void parse(char\* code) {

FILE\* fw=fopen("myscript.lx","w"); // Belirtilen ".ba" dosyasını ".(End of Line)" işaretinden sonra satırlara bölücek şekilde kodumuzu tasarladık.

int lineCount = 0;

char \*line[1000];

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

while (splitCode != NULL) {

line[lineCount++] = splitCode;

splitCode = strtok(NULL, ".");

}

for (int i = 0; i < lineCount; i++) {

int wordCount = 0;

char \*word[1000];

char \*splitLine = strtok(line[i], " \n\r[]\"");

while (splitLine != NULL) {

word[wordCount++] = splitLine;

splitLine = strtok(NULL, " \n\r");

}

if (wordCount == 0) { return; }

//Int => Identifier

if (!strcmp(word[0], "int")) { //Eğer bir satırın ilk kelimesi "int" şeklinde olursa buradaki if conditiona girecek

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

if (isVariable(word[1])) {

fprintf(fw,"Identifier %s \n", word[1]); //Devamında bigAdd Language'deki patternleri inceledik belirlenen pattern sonucunda int ile başlayan satır bir identifier ile devam etmeli.

variableList[vIndex++] = word[1];

integerList[iIndex++] = 0;

}

else {;

return 0;

}

if (wordCount == 2) {

fprintf(fw,"EndofLine\n");

}

}

//Move => Identifier => to =>Identifier

//Move => InsConstant => to =>Identifier

else if (!strcmp(word[0], "move")) { //If first letter of the line is "move". That code continious here.

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

int moveInt = 0;

if (isInteger(word[1])) {

fprintf(fw,"IntConstant %s \n", word[1]); // IntConstant or Identifier may come after the "move" keyword, and branches are provided accordingly

moveInt = atoi(word[1]);

}

else if (isIntegerVariable(word[1])) {

fprintf(fw,"Identifier %s \n", word[1]);

for (int i = 0; i < vIndex; i++) {

if (!strcmp(variableList[i], word[1])) {

moveInt = integerList[i];

break;

}

}

}

else {

return 0;

}

if (!strcmp(word[2], "to")) { //The 3rd command of the "Move" function must be the "to" keyword. Accordingly, the controls were provided.

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

}

else {;

return 0;

}

if (isIntegerVariable(word[3])) {

fprintf(fw,"Identifier %s \n", word[3]);

for (int i = 0; i < vIndex; i++) {

if (!strcmp(variableList[i], word[3])) {

integerList[i] = moveInt;

break;

}

}

}

else {

return 0;

}

if (wordCount == 4) {

fprintf(fw,"EndofLine\n"); //End of line with "End of Line" when the specified number of words is reached.

}

else {

return 0;

}

}

//Add => IntConstant => to => InsConstant

//Add => IntConstant => to => Identifier

//Add => Identifier => to => InsConstant

//Add => Identifier => to => Identifier

else if (!strcmp(word[0], "add")) { // The branches of the "Add" command are given above. Alternatives were developed according to all branching possibilities.

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

int addInt = 0;

if (isInteger(word[1])) {

fprintf(fw,"IntConstant %s.\n", word[1]);

addInt = atoi(word[1]);

}

else if (isIntegerVariable(word[1])) {

fprintf(fw,"Identifier %s \n", word[1]);

for (int i = 0; i < vIndex; i++) {

if (!strcmp(variableList[i], word[1])) {

addInt = integerList[i];

break;

}

}

}

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

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

}

if (isIntegerVariable(word[3])) {

fprintf(fw,"Identifier %s \n", word[3]);

for (int i = 0; i < vIndex; i++) {

if (!strcmp(variableList[i], word[3])) {

integerList[i] += addInt;

break;

}

}

}

if (isInteger(word[3])){

fprintf(fw,"IntConstant %s \n", word[3]);

}

if (wordCount == 4) {

fprintf(fw,"EndofLine\n");

}

}

//Sub => IntConstant => from => InsConstant

//Sub => IntConstant => from => Identifier

//Sub => Identifier => from => InsConstant

//Sub => Identifier => from => Identifier

else if (!strcmp(word[0], "sub")) { //Similar to the Add method, alternatives have been developed for branches.

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

int subInt = 0;

if (isInteger(word[1])) {

fprintf(fw,"IntConstant %s \n", word[1]);

subInt = atoi(word[1]);

}

else if (isIntegerVariable(word[1])) {

fprintf(fw,"Identifier %s \n", word[1]);

for (int i = 0; i < vIndex; i++) {

if (!strcmp(variableList[i], word[1])) {

subInt = integerList[i];

break;

}

}

}

else {

return 0;

}

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

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

}

else {

return 0;

}

if (isIntegerVariable(word[3])) {

fprintf(fw,"Identifier %s \n", word[3]);

for (int i = 0; i < vIndex; i++) {

if (!strcmp(variableList[i], word[3])) {

integerList[i] -= subInt;

break;

}

}

}

else if (isInteger(word[3])){

fprintf(fw,"IntConstant %s \n", word[3]);

}

else {

return 0;

}

if (wordCount == 4) {

fprintf(fw,"EndofLine\n");

}

}

else if (word[0][0] == ("%d",']')){

fprintf(fw,"CloseBlock \n");

}

//Out => IntConstant

//Out => StringConstant

//Out => Identifier => Seperator => Keyword newline

//Out => Keyword newline => Seperator => StringConstant => Seperator => Identifier

else if (!strcmp(word[0], "out")) { //Out bigAdd language's "print" function can therefore have many possible uses.

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

//Integer, Variable Or String

if (isInteger(word[1])) {

fprintf(fw,"IntConstant %s \n", word[1]); //We have developed their branching according to different situations.

}

else if (isIntegerVariable(word[1])) {

fprintf(fw,"Identifier %s \n", word[1]);

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

fprintf(fw,"Seperator \n");

if(!strcmp(word[3],"newline")){

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

if(wordCount==4){

fprintf(fw,"EndofLine \n");

}

}

}

}

else if (!strcmp(word[1],"newline")){

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

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

fprintf(fw,"Seperator \n");

fprintf(fw,"StringConstant %s %s",word[3],"\n");

if(!strcmp(word[4],",")){

fprintf(fw,"Seperator \n");

if(isVariable(word[5])){

fprintf(fw,"Identifier %s \n",word[5]);

if(wordCount == 6 ){

fprintf(fw,"EndofLine \n");

}

}

}

}

}

else {

fprintf(fw,"StringConstant %s\n", word[1]);

}

}

//Loop =>IntConstant => times => CodeBlock

//Loop =>Identifier => times => CodeBlock

else if (!strcmp(word[0], "loop")) { //If the first word of a split line is "loop", the if condition captures this condition.

fprintf(fw,"Keyword loop \n");

int loopInt = 0;

if (isInteger(word[1])) {

fprintf(fw,"IntConstant %s \n", word[1]);

loopInt = atoi(word[1]);

}

else if (isIntegerVariable(word[1])) {

fprintf(fw,"Identifier %s \n", word[1]);

for (int i = 0; i < vIndex; i++) {

if (!strcmp(variableList[i], word[1])) {

loopInt = integerList[i];

break;

}

}

}

else {

return 0;

}

//Times

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

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

if(word[3][0]==("%d",'[')){

fprintf(fw,"OpenBlock \n");

if(!strcmp(word[4],"out")){

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

if(isInteger(word[5])){

fprintf(fw,"IntConstant %s \n",word[5]);

seperatorControl(word[5]);

}

else if (isIntegerVariable(word[5])){

fprintf(fw,"Identifier %s \n", word[5] );

if(!strcmp(word[6],",")){

fprintf(fw,"Seperator \n");

if(!strcmp(word[7],"newline")){

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

if (wordCount == 8){

fprintf(fw,"EndofLine\n");

}

}

}

}

}

else if(!strcmp(word[4],"add")){

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

int addInt = 0 ;

if(isInteger(word[5])){

fprintf(fw,"IntConstant %s \n",word[5]);

addInt = atoi(word[5]);

}

else if (isIntegerVariable(word[5])){

fprintf(fw,"Identifier %s \n" , word[5]);

for(int i = 0 ; i < vIndex ; i++){

if(!strcmp(variableList[i],word[5])){

addInt = integerList[i];

break;

}

}

}

if (!strcmp(word[6],"to")){

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

}

if(isIntegerVariable(word[7])){

fprintf(fw,"Identifier %s \n",word[7]);

for(int i = 0 ; i<vIndex ; i++){

if(!strcmp(variableList[i],word[3])){

integerList[i] += addInt;

break;

}

}

}

if(isInteger(word[7])){

fprintf(fw,"IntConstant %s \n",word[7]);

}

if(wordCount == 8){

fprintf(fw,"EndofLine \n");

}

}

else if (!strcmp(word[4],"sub")){

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

int subInt = 0 ;

if(isInteger(word[4])){

fprintf(fw,"IntConstant %s",word[4]);

subInt = atoi(word[1]);

}

else if (isIntegerVariable(word[4])){

fprintf(fw,"Identifier %s \n",word[4]);

for (int i = 0 ; i<vIndex ; i++){

if(!strcmp(variableList[i],word[4])){

subInt = integerList[i];

break;

}

}

}

else {

return 0 ;

}

if(!strcmp(word[5],"from")){

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

}

else{

return 0 ;

}

if(isIntegerVariable(word[6])){

fprintf(fw,"Identifier %s \n",word[6]);

for(int i = 0 ; i<vIndex ; i++){

if(!strcmp(variableList[i],word[6])){

integerList[i] -= subInt;

break;

}

}

}

else if (isInteger(word[6])){

fprintf(fw,"IntConstant %s \n",word[6]);

}

if(wordCount == 7){

fprintf(fw,"EndofLine \n");

}

}

}

else if(!strcmp(word[3],"out")){

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

if(isInteger(word[4])){

fprintf(fw,"IntConstant %s \n",word[4]);

}

else if(isIntegerVariable(word[4])){

fprintf(fw,"Identifier %s \n",word[4]);

}

else {

fprintf(fw,"StringConstant %s \n",word[4]);

}

if(wordCount==4){

fprintf(fw,"EndofLine \n");

}

}

}

else {

return 0;

}

//CodeBlock

for (int i = 0; i < loopInt; i++) { //Since a new CodeBlock can be created in the loop, a recursive function to analyze the resulting CodeBlock

char newCode[100] = "";

for (int i = 3; i < wordCount; i++) {

strcat(newCode, " ");

strcat(newCode, word[i]);

}

strcat(newCode, ".");

parse(newCode);

}

}

}

fclose(fw);

}

char commentLine[100]; char a[100]; char b[100]; // The part that detects the comment line in bigAdd Language and allows that part of the code to be "ignore

void isComment(char\* str) {

FILE\* fw=fopen("myscript.lx","w");

bool trueComment = false;

int j;

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

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

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

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

substring(str, i + 1, j - i - 1, commentLine);

substring(str, 0, i, a);

substring(str, j + 1, strlen(str), b);

strcat(a, b);

strcpy(str, a);

trueComment = true;

isComment(str);

}

}

if (!trueComment) {

exit(0);

}

}

}

fclose(fw);

}

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

int length, i;

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

if (from > length) {

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;

}

int seperatorControl(char word[]){ // Eğer bir stringin sonunda "," varsa bunu algılayıp son virgülü iptal edip kelimenin doğru bir şekilde anlaşılmasını sağlayan fonksiyon

int uzunluk ;

if(word[(strlen(word)-1)]==','){

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

}

return word;

}

int main() {

FILE \*fp;

long lSize;

char \*code;

char command[40];

char fileName[40];

char data[40];

int i = 0 ;

char\* words[100];

printf("Enter Your Command : "); // Proje dökümanında lexical analyze işlemi "la myscript " komutuyla başlatılıyor bu isteğe göre.

gets(data);

char \*token = strtok(data," ");

while(token != NULL){ // Girilen iki kelime aralarındaki boşluğa göre iki parçaya ayırılıyor ve tanımlanan bir arrayin indislerine atanıyor

words[i++]=token;

token=strtok(NULL," ");

}

strcpy(command,words[0]); //Daha sonra bu indislerdeki kelimeler command ve fileName değişkenine atanıyor

strcpy(fileName,strcat(words[1],".ba")); //Dosya adına .ba uzantısı ekleniyor

if(!strcmp(command,"la")){ // Eğer ilk indisteki kelime doğru "la" komutuysa dosya adıyla birlikte işleme başlıyor.

fp = fopen(fileName,"rb");

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

}

else{

printf("Incorrect start command \n");

}

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

lSize = ftell(fp);

rewind(fp);

code = calloc(1, lSize + 1);

if (!code) fclose(fp), fputs("memory alloc fails", stderr), exit(1);

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

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

}

isComment(code);

parse(code);

}

## 2.b **Results**

