**Name: Mohammad Awais**

**Class: BSCS-8-A**

**CMS: 242554**

Compiler Construction Lab 2

Some source language specifications:

**Keywords:** break, case, char, const, continue, default, double, else, enum, extern, float, for, goto, if, int, long, return, short, static, struct, switch, void, while

**Arithmetic Operators:** + - \* / % ++ --

**Relational Operators:** == != > < >= <=

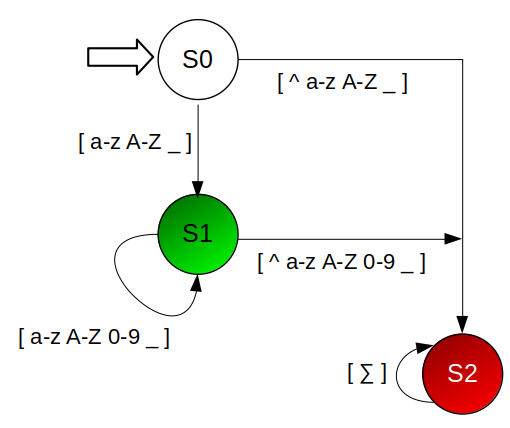
**Punctuators**: {} () [] = , . ; :

**Identifiers and Numbers** Floating Point and Integer Numbers

# Task 1: Comprehensive DFA

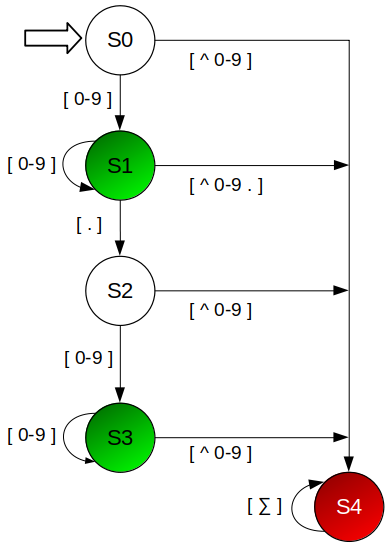
Keywords, Arithmetic Operators, Relational Operators & Punctuators would have simple table for comparison whereas DFA for **Identifiers** & **Numbers** ( Int & Float) would be required.

## Identifiers:



## Numbers:

**First Goal State is for Integer while Second is for Float.**



# Task 2: Skeleton Code in C++

## < Code – task2.cc >

#include <iostream>

#include <list>

#include <fstream>

#include <algorithm>

#include <bits/stdc++.h>

using namespace std;

bool inRange(int low,int high, int n){

if(n>=low && n<=high){

return true;

}

else

return false;

}

list<string> lexer\_sep(ifstream \*file);

void lexer\_seg(list<string> words);

*// Source Language Specificaitons*

string keywords[] = {"break","case","char","const","continue","default", "double", "else", "enum", "extern", "float", "for", "goto", "if", "int", "long", "return", "short", "static", "struct", "switch", "void", "while", "#include"};

string arithmetics[] = {"+","-","\*","/","%","++","--"};

string relationals[] = {"==","!=",">","<",">=","<="};

string punctuators[] = {"{","}","(",")","[","]","=",",",".",";",":"};

int identifier\_number(string word);

*// Reading File and lexer\_seg Analysis*

int main(){

string filename = "source.txt";

ifstream file(filename);

list<string> words = lexer\_sep(&file);

lexer\_seg(words);

file.close();

}

int identifier\_number(string word){

set<char> alphabets = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};

set <char> ALPHABETS;

for(char alph : alphabets){

ALPHABETS.insert(toupper(alph));

}

set<char> numbers = {'0','1','2','3','4','5','6','7','8','9'};

*// DFA CODE - IDENTIFIER*

int state = 0;

for (char c : word){

switch(state){

case 0:{

if(!(alphabets.count(c)||ALPHABETS.count(c)||c=='\_')){

state=2;

}

else{

state=1;

}

break;

}

case 1:{

if(alphabets.count(c)||ALPHABETS.count(c)||numbers.count(c)||c=='\_'){

}

else{

state=2;

}

break;

}

case 2:{

*//Failed State*

}

}

if(state==2){

break;

}

}

if(state==1){

return 1;

}

*// DFA CODE - NUMBER*

state = 0;

for (char c : word){

switch(state){

case 0:{

if(!(numbers.count(c))){

state=4;

}

else{

state=1;

}

break;

}

case 1:{

if(numbers.count(c)){

}

else if(c == '.'){

state = 2;

}

else{

state=4;

}

break;

}

case 2:{

if(!(numbers.count(c))){

state=4;

}

else{

state=3;

}

break;

}

case 3:{

if(numbers.count(c)){

}

else{

state=4;

}

break;

}

case 4:{

*//Failed State*

}

}

if(state==4){

break;

}

}

if(state==1 || state ==3){

return 2;

}

return 0;

}

*// FOR SEGMENTATION*

void lexer\_seg(list <string> words){

set <string> detect\_key;

set <string> detect\_punc;

set <string> detect\_ariths;

set <string> detect\_relate;

set <string> detect\_identif;

set <string> detect\_num;

set <string> detect\_others;

int segmented;

for (string word: words){

segmented = 0;

for(string target:keywords){

if(word.compare(target)==0){

detect\_key.insert(target);

segmented = 1;

break;

}

}

if(segmented==0){

for(string target:arithmetics){

if(word.compare(target)==0){

detect\_ariths.insert(target);

segmented = 1;

break;

}

}

}

if(segmented==0){

for(string target:relationals){

if(word.compare(target)==0 ){

detect\_relate.insert(target);

segmented = 1;

break;

}

}

}

if(segmented==0){

for(string target:punctuators){

if(word.compare(target)==0){

detect\_punc.insert(target);

segmented = 1;

break;

}

}

}

*// IDENTIFIER & NUMBER CHECK*

if(segmented==0){

int choice = identifier\_number(word);

if(choice==1){

detect\_identif.insert(word);

segmented = 1;

}

else if(choice == 2){

detect\_num.insert(word);

segmented = 1;

}

}

if(segmented==0){

detect\_others.insert(word);

}

}

cout<<"\n\t < Lexer Segmentation >\n\n";

cout<<"[+] KEYWORDS\n";

for (string word: detect\_key){

cout<<word<<"\n";

}

cout<<"\n[+] ARITHEMATIC OPERATORS\n";

for (string word: detect\_ariths){

cout<<word<<"\n";

}

cout<<"\n[+] RELATIONAL OPERATORS\n";

for (string word: detect\_relate){

cout<<word<<"\n";

}

cout<<"\n[+] PUNCTUATORS\n";

for (string word: detect\_punc){

cout<<word<<"\n";

}

cout<<"\n[+] IDENTIFIERS\n";

for (string word: detect\_identif){

cout<<word<<"\n";

}

cout<<"\n[+] NUMBERS\n";

for (string word: detect\_num){

cout<<word<<"\n";

}

cout<<"\n[+] OTHERS\n";

for (string word: detect\_others){

cout<<word<<"\n";

}

}

list<string> lexer\_sep(ifstream \*file){

string iLine;

list <string> sepline;

char c\_shad='\0';

int cState=0;

while(getline(\*file,iLine)){

string tempW="";

for (char c: iLine){

if( (c == ' ' || c == '\n' || c == '\t') && !(inRange(31,32,cState))){

if(!tempW.empty()){

sepline.push\_back(tempW);

tempW.clear();

}

cState = 0;

}

else if( (inRange(48,57,int(c)) || inRange(97,122,int(c)) || inRange(65,90,int(c)) || int(c)==95 || c == '.' || c=='#') && !(inRange(31,32,cState))){

if(!inRange(4,5,cState)){

if(!tempW.empty()){

sepline.push\_back(tempW);

tempW.clear();

}

if((inRange(48,57,int(c)))){

cState = 4;

}

else{

cState = 5;

}

}

else if(((inRange(48,57,int(c)) || c == '.' ) && cState ==4)){

}

else{

if(!tempW.empty() && cState ==4){

sepline.push\_back(tempW);

tempW.clear();

}

cState =5;

}

tempW+=c;

}

else if (c == '\'' || c == '"'){

if (inRange(31,32,cState)){

int check = 0;

if(c == '\'' && (cState==31)){

check = 1;

}

else if( c == '"' && (cState==32)){

check = 1;

}

tempW+=c;

if(check==1)

{

sepline.push\_back(tempW);

tempW.clear();

cState =-2;

}

}

else{

if(!tempW.empty()){

sepline.push\_back(tempW);

tempW.clear();

}

tempW+=c;

cState = (c == '\'')?31:32;

}

}

else if (inRange(31,32,cState)){

tempW+=c;

}

else if (!inRange(31,32,cState))

{

cState = -1;

if(!tempW.empty() && !(c\_shad == c && (c == '+' || c == '-' || c == '='))){

sepline.push\_back(tempW);

tempW.clear();

}

tempW+=c;

if(c\_shad == c){

c\_shad = '\0';

continue;

}

}

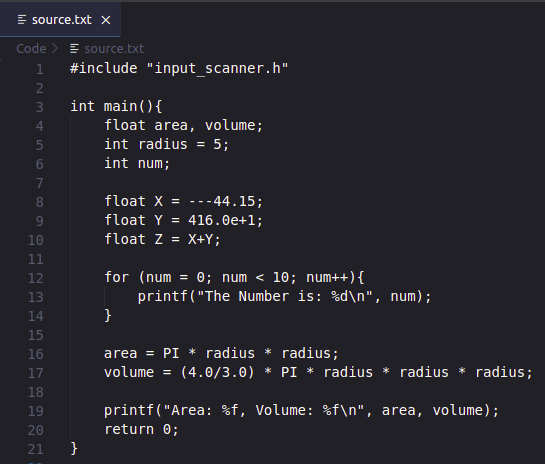
c\_shad = c;

}

}

return sepline;

}

**< Output – Screenshots >**

## 

# Task 3: **Single & Multi-Line Comment Removal**

## < Code – task3.cc >

#include <iostream>

#include <list>

#include <fstream>

#include <algorithm>

#include <bits/stdc++.h>

using namespace std;

void commentLess(ifstream \*file);

*// Reading File and lexer\_seg Analysis*

int main(){

string filename = "../Lab2Files/input\_scanner.c";

ifstream file(filename);

commentLess(&file);

file.close();

}

void commentLess(ifstream \*file){

ofstream file\_new("commentless.c");

string iLine;

string tempLine="";

char tempW='\0';

int commentMulti=0;

char c\_shadow='\0';

while(getline(\*file,iLine)){

for (char c: iLine){

if(commentMulti==1){

if(c=='/' && c\_shadow == '\*'){

if(tempW=='/'){

tempW='\0';

}

commentMulti = 0;

}

}

else if(c=='/' && c\_shadow!='/' ){

tempW='/';

}

else if(c=='/' && c\_shadow=='/'){

if(tempW == '/'){

tempW='\0';

}

tempLine+='\n';

break;

}

else if(c=='\*' && c\_shadow == '/'){

if(tempW=='/'){

tempW='\0';

}

commentMulti = 1;

}

else{

if(tempW == '/'){

tempLine+=tempW;

tempW='\0';

}

tempLine+=c;

}

c\_shadow = c;

}

if(commentMulti==1){

tempLine+='\n';

}

file\_new << tempLine;

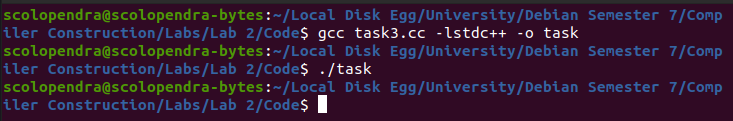
tempLine.clear();

}

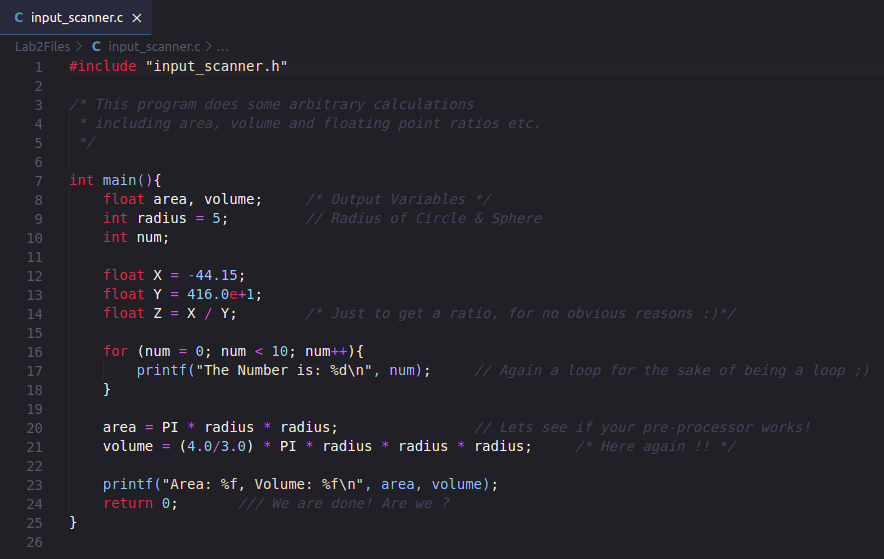
file\_new.close();

}

## < **Output - Screenshots** >



### **( Original )**



### **( Updated )**

