Name: Mohammad Awais
Class: BSCS-8-A
CMS: 242554

Compiler Construction Lab 2

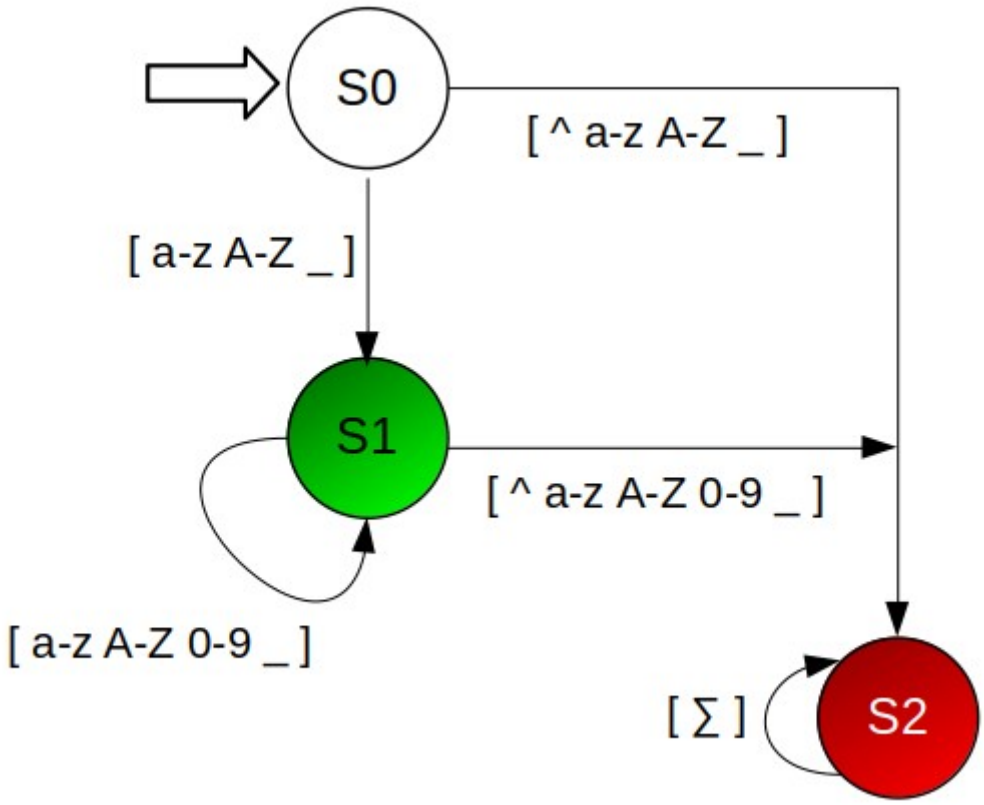
Some source language specifications:

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

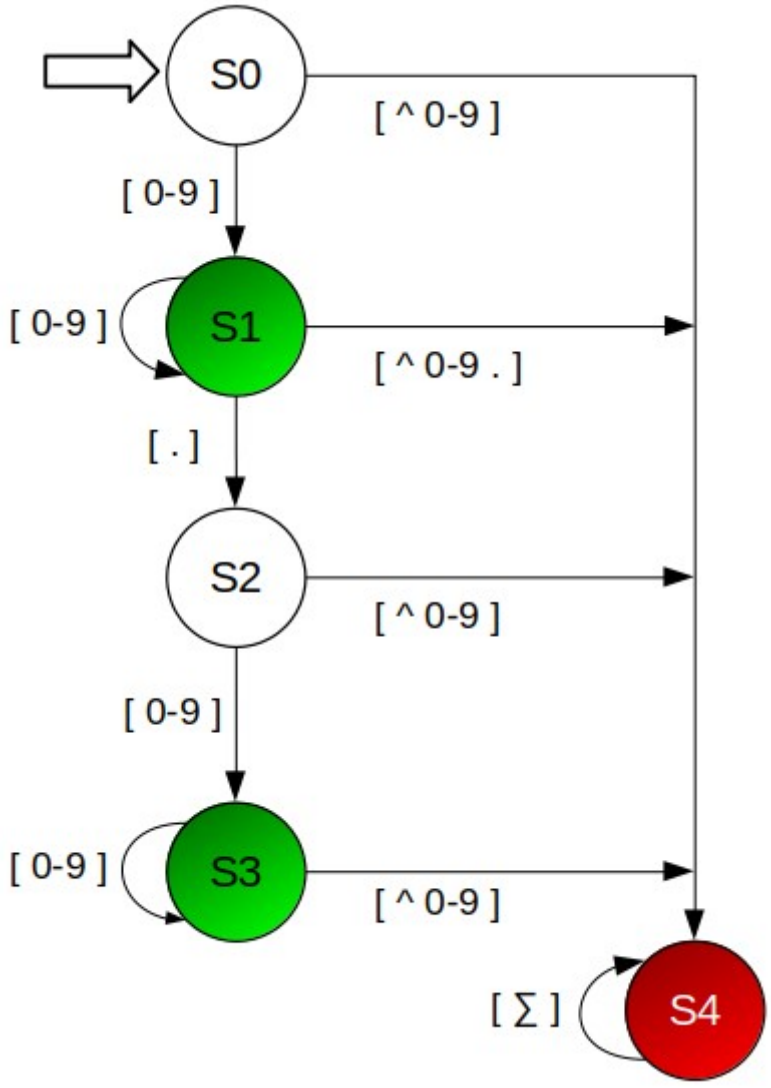
Keywords, Arithmetic Operators, Relational Operators & Punctuators would have simple table for comparison whereas DFA for **Identifiers** & **Numbers** (Int & Float) would be required.

- ### 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 >

source.txt

Code > source.txt

```
#include "input_scanner.h"

int main(){
    float area, volume;
    int radius = 5;
    int num;

    float X = ---44.15;
    float Y = 416.0e+1;
    float Z = X+Y;

    for (num = 0; num < 10; num++){
        printf("The Number is: %d\n", num);
    }

    area = PI * radius * radius;
    volume = (4.0/3.0) * PI * radius * radius * radius;

    printf("Area: %f, Volume: %f\n", area, volume);
    return 0;
}
```

```
scolopendra@scolopendra-bytes:~/Local Disk Egg/University/Debian Semester 7/Compiler Construction/Labs/Lab 2/Code$ gcc task2.cc -lstdc++ -o task
scolopendra@scolopendra-bytes:~/Local Disk Egg/University/Debian Semester 7/Compiler Construction/Labs/Lab 2/Code$ ./task

        < Lexer Segmentation >

[+] KEYWORDS
#include
float
for
int
return

[+] ARITHEMATIC OPERATORS
*
+
++
-
--
/

[+] RELATIONAL OPERATORS
<

[+] PUNCTUATORS
(
)
,
;
=
{
}
```

```
[+] IDENTIFIERS
PI
X
Y
Z
area
e
main
num
printf
radius
volume

[+] NUMBERS
0
1
10
3.0
4.0
416.0
44.15
5

[+] OTHERS
"Area: %f, Volume: %f\n"
"The Number is: %d\n"
scolopendra@scolopendra-bytes:~/Local Disk Egg/University/Debian Semester 7/Compiler Construction/Labs/Lab 2/Code$
```

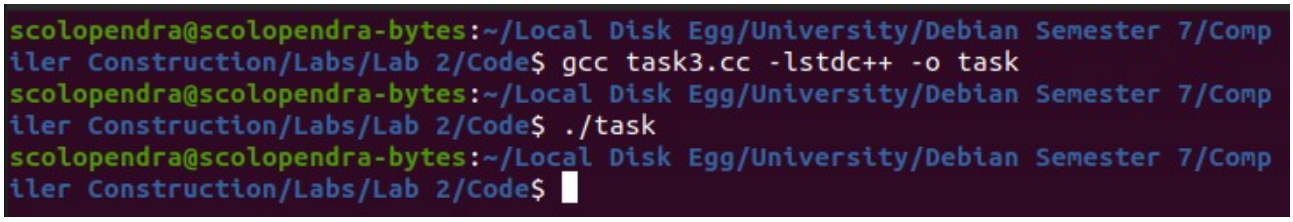
Task 3: Single & Multi-Line Comment Removal

< 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)

input_scanner.c

Lab2Files > input_scanner.c > ...

```c
#include "input_scanner.h"

/* This program does some arbitrary calculations
 * including area, volume and floating point ratios etc.
 */

int main(){
    float area, volume;     /* Output Variables */
    int radius = 5;         // Radius of Circle & Sphere
    int num;

    float X = -44.15;
    float Y = 416.0e+1;
    float Z = X / Y;        /* Just to get a ratio, for no obvious reasons :)*/

    for (num = 0; num < 10; num++){
        printf("The Number is: %d\n", num);     // Again a loop for the sake of being a loop ;)
    }

    area = PI * radius * radius;                // Lets see if your pre-processor works!
    volume = (4.0/3.0) * PI * radius * radius * radius;     /* Here again !! */

    printf("Area: %f, Volume: %f\n", area, volume);
    return 0;       /// We are done! Are we ?
}

```

(Updated)

commentless.c

Code > commentless.c > ...

```c
#include "input_scanner.h"



int main(){
    float area, volume;
    int radius = 5;
    int num;

    float X = -44.15;
    float Y = 416.0e+1;
    float Z = X / Y;

    for (num = 0; num < 10; num++){
        printf("The Number is: %d\n", num);
    }

    area = PI * radius * radius;
    volume = (4.0/3.0) * PI * radius * radius * radius;

    printf("Area: %f, Volume: %f\n", area, volume);
    return 0;
}

```