**Name: Mohammad Awais**

**Class: BSCS-8-A**

**CMS: 242554**

Compiler Construction Lab 3

# Tasks

## Code:

#include <iostream>

#include <list>

#include <fstream>

#include <algorithm>

#include <bits/stdc++.h>

#include <ctime>

using namespace std;

*// =============== SYMBOL-TABLE ========================*

*//---- some pre-requirements for Symbol Table*

class Dictionary{ *// CLass for Dictionary Objedct*

public:

string **value**;

int **hash**;

};

list<Dictionary> hash\_list;

int get\_hash(string value, int rank){ *// Used to get Hash to enter in Symbol-Table*

for (Dictionary d: hash\_list){

if(d.**value** == value){

return d.**hash**;

}

}

*// Calculated Random Hash -> Append List -> Return random hash*

Dictionary d;

d.**value** = value;

d.**hash** = rank;

hash\_list.push\_back(d);

return rank;

}

*// ----------- For Symbol Table, Primary Requirement*

class Token{

public:

string **name**;

string **value**;

int **hash**;

};

list<Token> symbol\_table;

void symbol\_table\_show(){

cout<<"\n\t |\t S Y M B O L \t T A B L E \t|\n";

cout<<"\n<TOKEN\_NAME>\t\t<TOKEN\_VALUE>\t\t<HASH\_VALUE>\n\n";

for (Token token: symbol\_table){

cout<<" "<<token.**name**<<"\t\t\t "<<token.**value**<<"\t\t\t "<<token.**hash**<<'\n';

}

}

*// -----------------------------------------------------*

bool inRange(int low,int high, int n){ *// USER defined self purpose*

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

return true;

}

else

return false;

}

string commentLess(ifstream \*file); *// Function to remove comments*

list<string> lexer\_sep(ifstream \*file); *// Function to Separate words from File*

void lexer\_seg(list<string> words); *// Function to Segment separated 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); *// DFA implementation to identify Identifier/Number(int/float)*

*// Reading File and lexer\_seg Analysis*

int main(){

string filename = "leapyear.c";

ifstream file(filename);

*// Separating Words*

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

*// Segmenting Words*

lexer\_seg(words);

*// Hash-value column*

cout<<"\n | HASH - VALUE | ~ ( Column Only )\n";

for (Dictionary d: hash\_list){

cout<<"\n "<<d.**hash**<<" - "<<d.**value**;

}

cout<<"\n";

*// Showing Symbol table*

symbol\_table\_show();

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){

string detect\_key = "KEY";

string detect\_punc = "PUNC";

string detect\_ariths = "OP-Ar";

string detect\_relate = "OP-Re";

string detect\_str = "STR";

string detect\_identif = "ID";

string detect\_num = "NUM";

string detect\_others = "OTHER";

int segmented;

Token miso;

int counter=1;

for (string word: words){

segmented = 0;

for(string target:keywords){

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

miso.**name** = detect\_key;

miso.**value** = word;

miso.**hash** = get\_hash(word,1000+counter);

counter++;

symbol\_table.push\_back(miso);

segmented = 1;

break;

}

}

if(segmented==0){

for(string target:arithmetics){

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

miso.**name** = detect\_ariths;

miso.**value** = word;

miso.**hash** = get\_hash(word,2000+counter);

counter++;

symbol\_table.push\_back(miso);

segmented = 1;

break;

}

}

}

if(segmented==0){

for(string target:relationals){

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

miso.**name** = detect\_relate;

miso.**value** = word;

miso.**hash** = get\_hash(word,3000+counter);

counter++;

symbol\_table.push\_back(miso);

segmented = 1;

break;

}

}

}

if(segmented==0){

for(string target:punctuators){

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

miso.**name** = detect\_punc;

miso.**value** = word;

miso.**hash** = get\_hash(word,4000+counter);

counter++;

symbol\_table.push\_back(miso);

segmented = 1;

break;

}

}

}

*// FOR STRINGS?*

if(segmented==0){

if(word[0]=='"'){

miso.**name** = detect\_str;

miso.**value** = word;

miso.**hash** = get\_hash(word,5000+counter);

counter++;

symbol\_table.push\_back(miso);

segmented = 1;

}

}

*// IDENTIFIER & NUMBER CHECK*

if(segmented==0){

int choice = identifier\_number(word);

if(choice==1){

miso.**name** = detect\_identif;

miso.**value** = word;

miso.**hash** = get\_hash(word,6000+counter);

counter++;

symbol\_table.push\_back(miso);

segmented = 1;

}

else if(choice == 2){

miso.**name** = detect\_num;

miso.**value** = word;

miso.**hash** = get\_hash(word,7000+counter);

counter++;

symbol\_table.push\_back(miso);

segmented = 1;

}

}

if(segmented==0 && word!="\r"){

miso.**name** = detect\_others;

miso.**value** = word;

miso.**hash** = get\_hash(word,8000+counter);

counter++;

symbol\_table.push\_back(miso);

}

}

}

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

string iLine = commentLess(file);

list <string> sepline;

char c\_shad='\0';

int cState=0;

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;

}

string commentLess(ifstream \*file){

string commentless;

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';

}

commentless+=tempLine;

tempLine.clear();

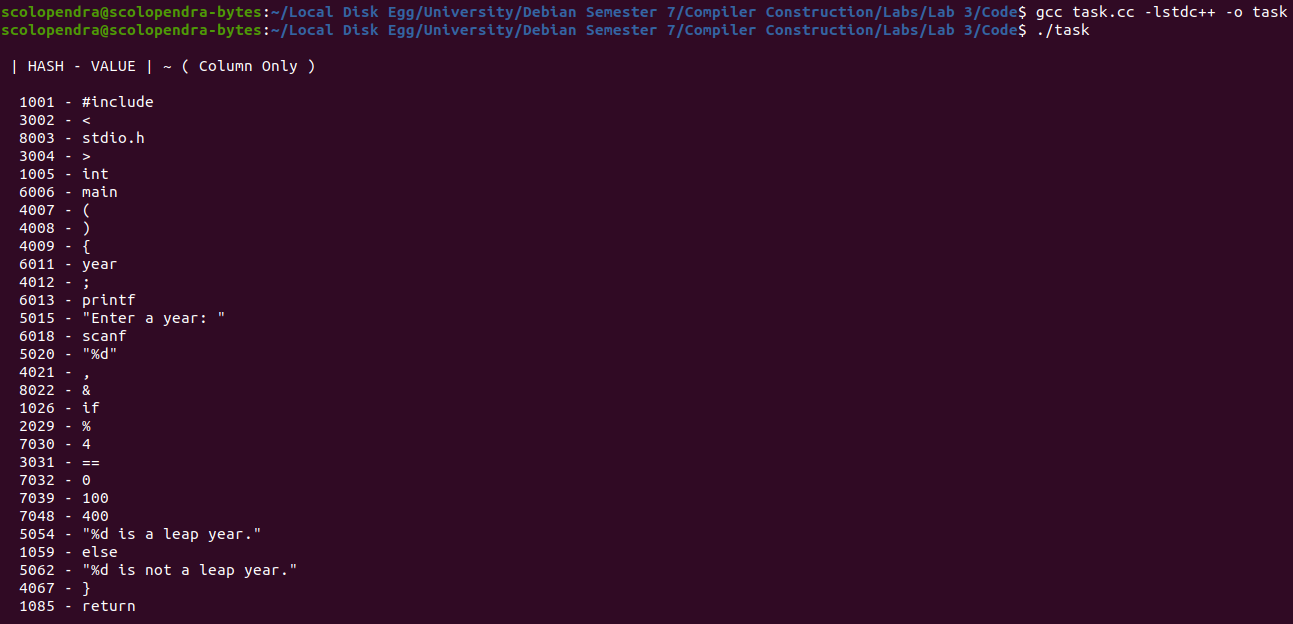
}

return commentless;

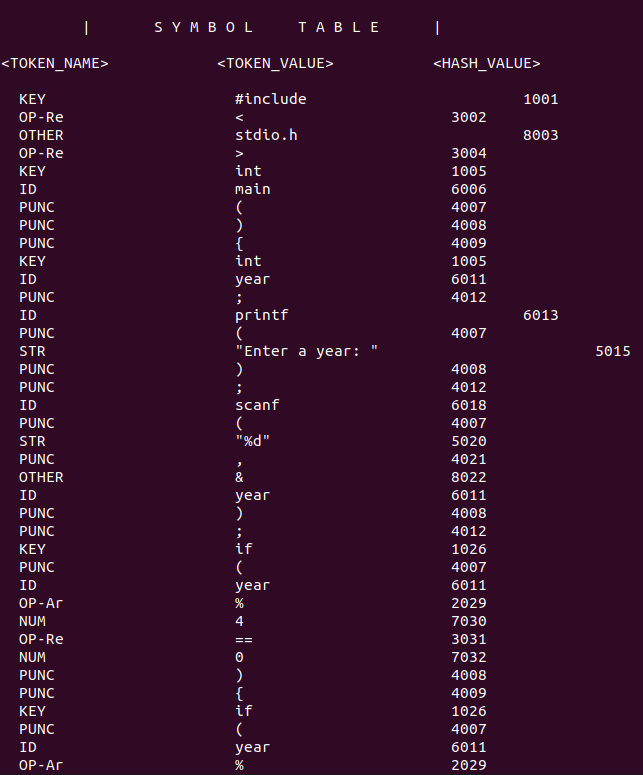
}

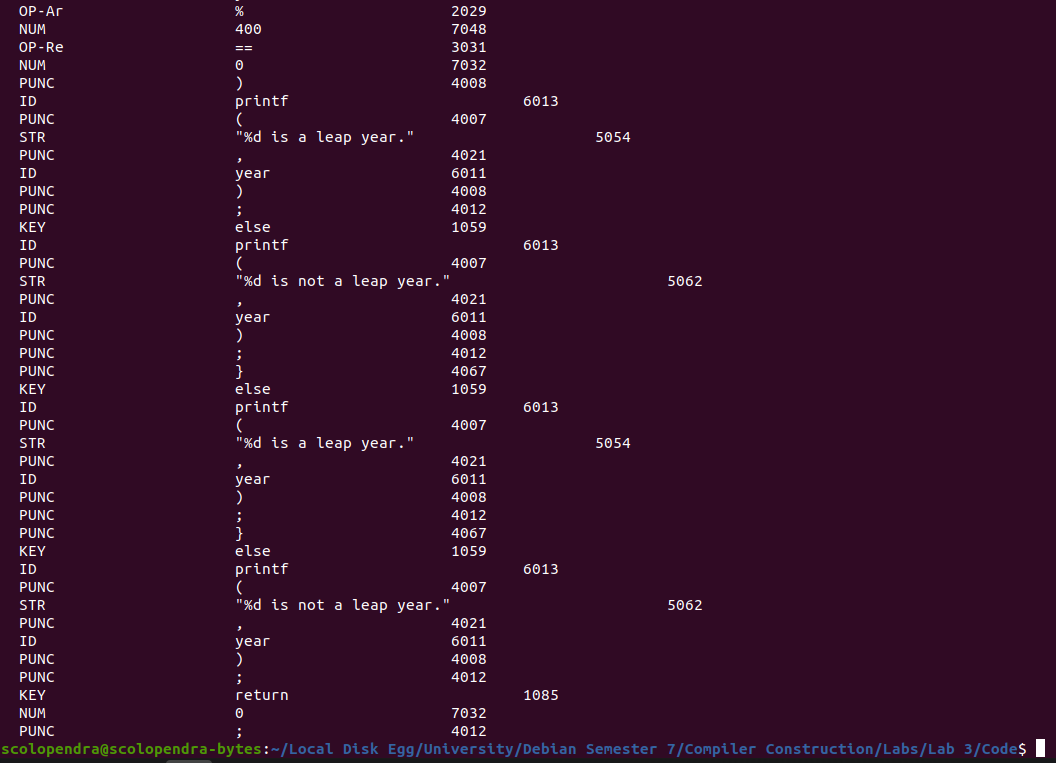
### Output:

### **Hash-Value Column**



### **Symbol Table ( Continued )**





### **Referenced File**

