

**Ahsanullah University of Science & Technology**

**Department of Computer Science & Engineering**

**Course No : CSE4130**

**Course Title : Formal Languages and Compilers Lab**

**Assignment No : 03**

**Date of Performance : 05/01/2022**

**Date of Submission : 19/01/2022**

**Submitted To : Mr. Aminur Rahman & Mr. Al Hasib Mahamud**

**Submitted By-**

**Group : B1**

**Name : Debopriya Deb Roy**

**Id : 190104065**

**Section : B**

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#include <stdlib.h>

#define TABLE\_SIZE 10

#define MAX\_NAME 256

typedef struct {

char name[MAX\_NAME];

int sl\_no;

} Name;

typedef struct {

char id\_type[MAX\_NAME];

int sl\_no;

} ID\_type;

typedef struct {

char data\_type[MAX\_NAME];

int sl\_no;

} DATA\_type;

typedef struct {

char scope[MAX\_NAME];

int sl\_no;

} SCOPE;

typedef struct {

char value[MAX\_NAME];

int sl\_no;

} VALUE;

Name\* name\_table[TABLE\_SIZE];

ID\_type\* id\_type[TABLE\_SIZE];

DATA\_type\* data\_type[TABLE\_SIZE];

SCOPE\* scope[TABLE\_SIZE];

VALUE\* value[TABLE\_SIZE];

int hash(char\* name)

{

int hash\_value = 0,i;

for ( i = 0; i < MAX\_NAME; i++) {

hash\_value += name[i];

hash\_value = (hash\_value \* name[i]) % 10;

}

return hash\_value;

}

void free\_table()

{

int i;

for ( i = 0; i < TABLE\_SIZE; i++) {

name\_table[i] = NULL;

id\_type[i] = NULL;

data\_type[i] = NULL;

scope[i] = NULL;

value[i] = NULL;

}

}

int sl\_number = 0;

int entry\_in\_symol\_table = 7;

void display()

{

printf("\nStarting display\n");

while (true)

{

sl\_number++;

int i;

for ( i = 0; i < TABLE\_SIZE; i++)

{

if (name\_table[i] != NULL) {

if (name\_table[i]->sl\_no == sl\_number)

{

printf("%d\t", sl\_number);

printf("%s\t", name\_table[i]->name);

}

if (id\_type[i]->sl\_no == sl\_number)

{

printf("%s\t", id\_type[i]->id\_type);

}

if (data\_type[i]->sl\_no == sl\_number)

{

printf("%s\t", data\_type[i]->data\_type);

}

if (scope[i]->sl\_no == sl\_number)

{

printf("%s\t", scope[i]->scope);

}

if (value[i]->sl\_no == sl\_number)

{

if ((strcmp(value[i]->value, "null"))) {

printf("%s\t", value[i]->value);

}

printf("\n");

break;

}

}

}

if (sl\_number == entry\_in\_symol\_table) {

break;

}

}

printf("\nEnding display\n");

}

Name\* lookup(char\* name)

{

int index = hash(name);

if (name\_table[index] != NULL && !(strcmp(name\_table[index]->name, name)))

{

return name\_table[index];

}

else if (name\_table[index] != NULL) {

int i;

for ( i = 0; i < TABLE\_SIZE; i++) {

int temp = (i + index) % TABLE\_SIZE;

if (name\_table[temp] != NULL) {

if (name\_table[temp]->name == name) {

return name\_table[temp];

}

}

}

return NULL;

}

else {

return NULL;

}

}

int lookup\_func(char\* name, char\* sc)

{

int index = hash(name);

if (name\_table[index] != NULL && !(strcmp(name\_table[index]->name, name)) && !(strcmp(scope[index]->scope, sc)))

{

return index;

}

else if (name\_table[index] != NULL) {

int i;

for ( i = 0; i < TABLE\_SIZE; i++) {

int temp = (i + index) % TABLE\_SIZE;

if (name\_table[temp] != NULL) {

if (!(strcmp(name\_table[temp]->name, name)) && !(strcmp(scope[temp]->scope, sc))) {

return temp;

}

}

}

return 1111;

}

else {

return 1111;

}

}

bool insert(Name\* n, DATA\_type\* dt, SCOPE\* sc)

{

if (n == NULL)

{

return false;

}

if (lookup\_func(n->name, sc->scope) == 1111 && name\_table[hash(n->name)] != NULL)

{

int index = hash(n->name);

int i;

for ( i = 0; i < TABLE\_SIZE; i++) {

int temp = (i + index) % TABLE\_SIZE;

if (name\_table[temp] == NULL) {

name\_table[temp] = n;

data\_type[temp] = dt;

scope[temp] = sc;

return true;

}

}

return false;

}

else {

int index = hash(n->name);

name\_table[index] = n;

data\_type[index] = dt;

scope[index] = sc;

return true;

}

}

int num\_rec(char\* lex)

{

int s, i, l;

i = 0;

s = 0;

if (isdigit(lex[i])) {

s = 1;

i++;

}

else if (lex[i] == '.') {

s = 2;

i++;

}

else {

s = 0;

}

l = strlen(lex);

if (s == 1) {

for (; i < l; i++)

{

if (isdigit(lex[i])) {

s = 1;

}

else if (lex[i] == '.') {

s = 2;

i++;

break;

}

else {

s = 0;

break;

}

}

}

if (s == 2)

{

if (isdigit(lex[i])) {

s = 3;

i++;

}

else {

s = 0;

}

}

if (s == 3) {

for (; i < l; i++)

{

if (isdigit(lex[i])) {

s = 3;

}

else {

s = 0;

break;

}

}

}

if (s == 3) {

s = 1;

}

return s;

}

int id\_rec(char\* lex)

{

int i, s, l;

i = 0;

s = 0;

if ((lex[i] >= 'a' && lex[i] <= 'z') || (lex[i] >= 'A' && lex[i] <= 'Z')) {

s = 1;

}

else if (lex[i] == '\_') {

s = 1;

}

else {

s = 0;

}

l = strlen(lex);

if (s == 1)

{

for (; i < l; i++)

{

if (isdigit(lex[i])) {

}

else if (lex[i] == '\_') {

}

else if ((lex[i] >= 'a' && lex[i] <= 'z') || (lex[i] >= 'A' && lex[i] <= 'Z')) {

}

else {

s = 0;

break;

}

}

}

return s;

}

int key\_rec(char\* lex)

{

int s = 0;

if (!(strcmp(lex, "char")) || !(strcmp(lex, "int")) ||

!(strcmp(lex, "void")) || !(strcmp(lex, "double")) ||

!(strcmp(lex, "return")) || !(strcmp(lex, "float")) ||

!(strcmp(lex, "if")) || !(strcmp(lex, "else"))) {

s = 1;

}

return s;

}

int sep\_rec(char\* lex)

{

int s = 0;

if (!(strcmp(lex, ",")) || !(strcmp(lex, ";")) || !(strcmp(lex, "\'"))) {

s = 1;

}

return s;

}

int par\_rec(char\* lex)

{

int s = 0;

if (!(strcmp(lex, "(")) || !(strcmp(lex, ")")) || !(strcmp(lex, "{")) || !(strcmp(lex, "}"))) {

s = 1;

}

return s;

}

int op\_rec(char\* lex)

{

int s = 0;

if (!(strcmp(lex, "=")) || !(strcmp(lex, "+")) || !(strcmp(lex, "-"))

|| !(strcmp(lex, "\*")) || !(strcmp(lex, "/")) || !(strcmp(lex, "<="))) {

s = 1;

}

return s;

}

void catagorized(FILE\* p, char\* lexeme, int length, FILE \*p1)

{

char lex[10];

int i;

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

lex[i] = lexeme[i];

}

lex[length] = '\0';

if (key\_rec(lex)) {

fputc('[', p);

fputs("kw", p);

fputc(' ', p);

fputs(lex, p);

fputc(']', p);

fputc('[', p1);

fputs(lex, p1);

fputc(']', p1);

}

else {

if (id\_rec(lex)) {

fputc('[', p);

fputs("id", p);

fputc(' ', p);

fputs(lex, p);

fputc(']', p);

fputc('[', p1);

fputs("id", p1);

fputc(' ', p1);

fputs(lex, p1);

fputc(']', p1);

}

else {

if (num\_rec(lex)) {

fputc('[', p);

fputs("num", p);

fputc(' ', p);

fputs(lex, p);

fputc(']', p);

fputc('[', p1);

fputs(lex, p1);

fputc(']', p1);

}

else {

if (sep\_rec(lex)) {

fputc('[', p);

fputs("sep", p);

fputc(' ', p);

fputs(lex, p);

fputc(']', p);

fputc('[', p1);

fputs(lex, p1);

fputc(']', p1);

}

else {

if (op\_rec(lex)) {

fputc('[', p);

fputs("op", p);

fputc(' ', p);

fputs(lex, p);

fputc(']', p);

fputc('[', p1);

fputs(lex, p1);

fputc(']', p1);

}

else {

if (par\_rec(lex)) {

fputc('[', p);

fputs("par", p);

fputc(' ', p);

fputs(lex, p);

fputc(']', p);

fputc('[', p1);

fputs(lex, p1);

fputc(']', p1);

}

else {

fputc('[', p);

fputs("unkn", p);

fputc(' ', p);

fputs(lex, p);

fputc(']', p);

fputc('[', p1);

fputs("unkn", p1);

fputc(' ', p1);

fputs(lex, p1);

fputc(']', p1);

printf("\nError : [ %s", lex);

printf(" ]\n");

}

}

}

}

}

}

}

int serial\_number = 0;

void generation\_of\_symbol\_table(char \*lexeme, char \*d, char \*s, int len)

{

char id[10];

int i, j;

for (i = 3, j = 0; i < len; i++, j++) {

id[j] = lexeme[i];

}

id[j] = '\0';

if (lookup\_func(id, s) == 1111) {

Name\* name = (Name\*)malloc(sizeof(Name));

memcpy(name->name, id, sizeof(id));

name->sl\_no = ++serial\_number;

DATA\_type\* data = (DATA\_type\*)malloc(sizeof(DATA\_type));

memcpy(data->data\_type, d, sizeof(d) + sizeof(d));

data->sl\_no = serial\_number;

SCOPE\* scp = (SCOPE\*)malloc(sizeof(SCOPE));

memcpy(scp->scope, s, sizeof(s) + sizeof(s));

scp->sl\_no = serial\_number;

insert(name, data, scp);

}

/\*else if (lookup(id) != NULL) {

int index = hash(id);

if (name\_table[index] != NULL && !(strcmp(name\_table[index]->name, id)))

{

if (!(strcmp(scope[index]->scope, s))) {

}

else {

Name\* name = (Name\*)malloc(sizeof(Name));

memcpy(name->name, id, sizeof(id));

name->sl\_no = ++serial\_number;

DATA\_type\* data = (DATA\_type\*)malloc(sizeof(DATA\_type));

memcpy(data->data\_type, d, sizeof(d) + sizeof(d));

data->sl\_no = serial\_number;

SCOPE\* scp = (SCOPE\*)malloc(sizeof(SCOPE));

memcpy(scp->scope, s, sizeof(s) + sizeof(s));

scp->sl\_no = serial\_number;

insert(name, data, scp);

}

}

else if (name\_table[index] != NULL) {

for (int i = 0; i < TABLE\_SIZE; i++) {

int temp = (i + index) % TABLE\_SIZE;

if (name\_table[temp] != NULL) {

if (name\_table[temp] == id) {

if (!(strcmp(scope[temp]->scope, s))) {

}

else {

Name\* name = (Name\*)malloc(sizeof(Name));

memcpy(name->name, id, sizeof(id));

name->sl\_no = ++serial\_number;

DATA\_type\* data = (DATA\_type\*)malloc(sizeof(DATA\_type));

memcpy(data->data\_type, d, sizeof(d) + sizeof(d));

data->sl\_no = serial\_number;

SCOPE\* scp = (SCOPE\*)malloc(sizeof(SCOPE));

memcpy(scp->scope, s, sizeof(s) + sizeof(s));

scp->sl\_no = serial\_number;

insert(name, data, scp);

}

}

}

}

}

}\*/

}

void update\_symbol\_table(char \*lexeme, char \*number, char \*id\_t, char \*sc, int len)

{

char id\_st[10];

int i, j;

for (i = 3, j = 0; i < len; i++, j++) {

id\_st[j] = lexeme[i];

}

id\_st[j] = '\0';

int index = hash(id\_st);

if (name\_table[index] != NULL && !(strcmp(name\_table[index]->name, id\_st)) && !(strcmp(scope[index]->scope, sc)))

{

ID\_type\* id = (ID\_type\*)malloc(sizeof(ID\_type));

memcpy(id->id\_type, id\_t, sizeof(id\_t) + sizeof(id\_t));

id->sl\_no = name\_table[index]->sl\_no ;

id\_type[index] = id;

VALUE\* v = (VALUE\*)malloc(sizeof(VALUE));

memcpy(v->value, number, sizeof(number) + sizeof(number));

v->sl\_no = name\_table[index]->sl\_no;

value[index] = v;

}

else if (name\_table[index] != NULL) {

int i;

for ( i = 0; i < TABLE\_SIZE; i++) {

int temp = (i + index) % TABLE\_SIZE;

if (name\_table[temp] != NULL && !(strcmp(name\_table[temp]->name, id\_st)) && !(strcmp(scope[temp]->scope, sc))) {

ID\_type\* id = (ID\_type\*)malloc(sizeof(ID\_type));

memcpy(id->id\_type, id\_t, sizeof(id\_t) + sizeof(id\_t));

id->sl\_no = name\_table[temp]->sl\_no;

id\_type[temp] = id;

VALUE\* v = (VALUE\*)malloc(sizeof(VALUE));

memcpy(v->value, number, sizeof(number) + sizeof(number));

v->sl\_no = name\_table[temp]->sl\_no;

value[temp] = v;

}

}

}

else {

return;

}

}

int modify(char \*name)

{

int i;

for ( i = 0; i < TABLE\_SIZE; i++)

{

if (name\_table[i] != NULL) {

if (!(strcmp(name\_table[i]->name, name)))

{

return name\_table[i]->sl\_no;

}

}

}

}

int main()

{

FILE\* p1, \* p2, \* p3, \* p4;

char c;

char c2;// = c; //for no space after if

p1 = fopen("input1.c", "r");

p2 = fopen("input2.txt", "w");

while ((c = fgetc(p1)) != EOF) {

if (c == '\n') {

c = fgetc(p1);

if (c == ' ' || c == '\n') {

while ((c = fgetc(p1)) == ' ') {}

if ((c != '(' && c != ')' && c != '{' && c != '}') && (c2 != '(' && c2 != ')' && c2 != '{' && c2 != '}')) {

fputc(' ', p2); // for separating ==> "else c"

}

if (c == '/') {

c = fgetc(p1);

if (c == '/') {

while ((c = fgetc(p1)) != '\n') {}

}

else if (c == '\*') {

while ((c = fgetc(p1)) != '/') {}

}

}

else {

fputc(c, p2);

}

}

else if (c != '\n') {

if (c == '/') {

char c1 = c;

c = fgetc(p1);

if (c == '/') {

while ((c = fgetc(p1)) != '\n') {}

}

else if (c == '\*') {

while ((c = fgetc(p1)) != '/') {}

}

else {

fputc(c1, p2);

fputc(c, p2);

}

}

else {

fputc(c, p2);

}

}

}

else if (c == ' ') {

fputc(c, p2);

while ((c = fgetc(p1)) == ' ') {}

fputc(c, p2);

}

else if (c == '/') {

char c1 = c;

c = fgetc(p1);

if (c == '/') {

while ((c = fgetc(p1)) != '\n') {}

}

else if (c == '\*') {

while ((c = fgetc(p1)) != '/') {}

}

else {

fputc(c1, p2);

fputc(c, p2);

}

}

else if (c == '"') {

fputc(c, p2);

while ((c = fgetc(p1)) != '"')

{

fputc(c, p2);

}

fputc(c, p2);

}

else {

fputc(c, p2);

c2 = c;

}

}

fclose(p1);

fclose(p2);

p2 = fopen("input2.txt", "r");

p3 = fopen("output1.txt", "w");

bool flag = false;

bool space = false;

bool just\_one\_space = false;

bool already\_space = false;

while ((c = fgetc(p2)) != EOF)

{

if (c == ' ') {

space = true;

just\_one\_space = true;

}

if (c == ';' || c == ',' || c == '=' || c == '+' || c == '\*' ||

c == '-' || c == '/' || c == '(' || c == ')' || c == '<' ||

c == '\'' || c == '{' || c == '}') {

if (c == '<') {

c = fgetc(p2);

if (c == '=') {

fputc(' ', p3);

fputc('<', p3);

fputc('=', p3);

fputc(' ', p3);

}

else {

fputc(' ', p3);

fputc(c, p3);

fputc(' ', p3);

}

}

else if (c == '{' || c == '}') {

fputc(c, p3);

fputc(' ', p3);

}

else {

if (c == '\'') {

if (flag) {

fputc(' ', p3);

fputc(c, p3);

flag = false;

}

else {

fputc(c, p3);

fputc(' ', p3);

flag = true;

}

}

else {

if (!space) {

fputc(' ', p3);

fputc(c, p3);

fputc(' ', p3);

already\_space = true;

space = true;

just\_one\_space = true;

}

else {

if (just\_one\_space && c == ';') {

fputc(c, p3);

fputc(' ', p3);

just\_one\_space = false;

}

else {

fputc(c, p3);

}

space = false;

}

}

}

}

else {

if (c == ' ' && !already\_space) {

fputc(c, p3);

already\_space = false;

}

else if (c != ' ') {

fputc(c, p3);

already\_space = false;

just\_one\_space = false;

space = false;

}

if (space && (((c >= 'a') && (c <= 'z')) || isdigit(c))) {

space = false;

just\_one\_space = false;

}

}

}

fclose(p2);

fclose(p3);

// Second Step

p2 = fopen("output1.txt", "r");

p3 = fopen("output2.txt", "w");

p4 = fopen("output3.txt", "w");

printf("\n");

while (true) {

char lexeme[10];

int index = 0;

if ((c = fgetc(p2)) == EOF) {

break;

}

else {

lexeme[index++] = c;

}

c = fgetc(p2);

if (c == EOF) {

break;

}

while ((c != ' ')) {

lexeme[index++] = c;

c = fgetc(p2);

}

catagorized(p3, lexeme, index, p4);

}

fclose(p2);

fclose(p3);

fclose(p4);

printf("\nStep 1 Output: \n");

p2 = fopen("output3.txt", "r");

while ((c = fgetc(p2)) != EOF) {

printf("%c", c);

}

fclose(p2);

//Step 02 : Symbol table generation

free\_table();

p1 = fopen("output3.txt", "r");

char scope[7] = "global";

char find\_scope[10];

char store\_lexeme\_name[10];

int update\_table\_len = 0;

bool identifier = false;

bool identifier\_for\_scope = false;

bool check\_digit = true;

while ((c = fgetc(p1)) != EOF) {

if (c == '[') {

char lexeme[10];

char data\_type[10];

int i = 0;

while (c != ']') {

c = fgetc(p1);

if (c == ' ') {

identifier = true;

}

if (c != ']') {

lexeme[i++] = c;

}

}

if (identifier) {

int j;

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

store\_lexeme\_name[j] = lexeme[j];

}

store\_lexeme\_name[i] = '\0';

update\_table\_len = i;

generation\_of\_symbol\_table(lexeme, data\_type, scope, i); //pass scope

identifier = false;

identifier\_for\_scope = true;

}

else {

int j;

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

data\_type[j] = lexeme[j];

}

data\_type[i] = '\0';

if (key\_rec(data\_type)) {

//it is a data type

}

else {

if (!(strcmp(data\_type, ";"))) {

identifier\_for\_scope = false;

}

else if (!(strcmp(data\_type, "="))) {

identifier\_for\_scope = false;

char num[10];

i = 0;

c = fgetc(p1);

while (c != ']') {

c = fgetc(p1);

if (c != ']') {

num[i++] = c;

}

}

num[i] = '\0';

if (num\_rec(num)) {

update\_symbol\_table(store\_lexeme\_name, num, "var", scope, update\_table\_len);

}

else {

update\_symbol\_table(store\_lexeme\_name, "null", "var", scope, update\_table\_len);

}

}

else if (!(strcmp(data\_type, "(")) && identifier\_for\_scope == true) {

update\_symbol\_table(store\_lexeme\_name, "null", "func", scope, update\_table\_len);

int k, j;

for (k = 0, j = 3; j < update\_table\_len; j++, k++) {

scope[k] = store\_lexeme\_name[j];

}

scope[k] = '\0';

identifier\_for\_scope = false;

}

else if (!(strcmp(data\_type, "{")) && identifier\_for\_scope == true) {

identifier\_for\_scope = false;

update\_symbol\_table(store\_lexeme\_name, "null", "var", scope, update\_table\_len);

}

else if (!(strcmp(data\_type, "}"))) {

scope[0] = 'g';scope[1] = 'l';scope[2] = 'o';scope[3] = 'b';

scope[4] = 'a';scope[5] = 'l';scope[6] = '\0';

}

}

}

}

}

fclose(p1);

display();

//step 04 : Modified token stream for Syntax analysis

p1 = fopen("output3.txt", "r");

p2 = fopen("output4.txt", "w");

while ((c = fgetc(p1)) != EOF) {

fputc(c, p2);

if (c == ' ') {

int i = 0;

char lex[10];

while (c != ']')

{ c = fgetc(p1);

if (c != ']')

{

lex[i++] = c;

} }

lex[i] = '\0';

int x = modify(lex);

char buffer[10];

\_itoa(x, buffer, 10);

fputs(buffer, p2);

fputc(c, p2);

}

}

fclose(p1);

fclose(p2);

printf("\nStep 4 Output: \n");

p2 = fopen("output4.txt", "r");

while ((c = fgetc(p2)) != EOF) {

printf("%c", c);

}

fclose(p2);

return 0;

}