Name : Rajvardhan Reddy
Reg No : 180905093
Sec : B
Roll No : 19

CD LAB4 : CONSTRUCTION OF SYMBOL TABLE

P1) Using getNextToken( ) implemented in Lab No 3, design a Lexical Analyzer to implement local and global symbol table to store tokens for identifiers using array of structure.

input.c (used for parsing) :

```
#include<stdio.h>
int main(){
int p,q,m,n;
printf("Enter the dimensions of the first matrix: ");
scanf("%d%d",&p,&q);
printf("Enter the dimensions of the second matrix: ");
scanf("%d%d",&m,&n);
if(m!=q){
printf("The matrices are unsuitable for multiplication");
return 0;
}
int size1 = p*q*sizeof(int);
int size2 = m*n*sizeof(int);
int *arr1 = (int*)malloc(size1);int *arr2 = (int*)malloc(size2);
printf("Enter the elements of the first array\n");
int i,j,k;
for (i=0;i<p;i++){
for(j=0;j<q;j++){
scanf("%d",arr1+q*i+j);
}
}
printf("Enter the elements of the second array\n");
for (i=0;i<m;i++){
for(j=0;j<n;j++){
scanf("%d",arr2+n*i+j);
}
}
int size3 = p*n*sizeof(int);
int *res = (int*)malloc(size3);
```

```
int sum = 0;
for(i=0;i<p;i++){
for(j=0;j<n;j++){
for(k=0;k<m;k++){
sum+= arr1[i*m+k] * arr2[k*n+j];
}
res[i*n+j] = sum;
sum=0;
}
}
printf("The result is\n");
for(i=0;i<p;i++){
for(j=0;j<n;j++){
printf("%d ",*(res+n*i+j));
}printf("\n");
}
return 0;
}
```

lab4_p1.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
const char *keywords[] = {
"auto","double","int","struct","break","else","long","switch","case","enum","register"
,
"typedef","char","extern","return","union","continue","for","signed","void","do","if",
"static","while","default","goto","sizeof","volatile","const","float","short","unsigned"
,
"printf","scanf","true","false"
};
const char *datypes[]={"int","char","void","float","bool"};
int isdtype(char *w)
{
int i;
for(i=0;i<sizeof(datypes)/sizeof(char*);i++)
{
if(strcmp(w,datypes[i])==0)
{
return 1;
}
}
```

```
return 0;
}
int isKeyword(char *w){
int i;
for(i=0;i<sizeof(keywords)/sizeof(char*);i++)
{
if(strcmp(w,keywords[i])==0)
{
return 1;
}
}
return 0;
}
struct token
{
char lexeme[128];
unsigned int row,col;
char type[64];
};
struct sttable
{
int sno;
char lexeme[128];
char dtype[64];
char type[64];
int size;
};
int findTable(struct sttable *tab,char *nam,int n)
{
int i=0;
for(i=0;i<n;i++)
{
if(strcmp(tab[i].lexeme,nam)==0)
{
return 1;
}
}
return 0;
}
struct sttable fillTable(int sno,char *lexn,char *dt,char *t,int s)
{
struct sttable tab;
tab.sno=sno;
strcpy(tab.lexeme,lexn);
strcpy(tab.dtype,dt);strcpy(tab.type,t);
```

```
tab.size=s;
return tab;
}
void printTable(struct sttable *tab,int n)
{
for(int i=0;i<n;i++)
{
printf("%d %s %s %s %d \
n",tab[i].sno,tab[i].lexeme,tab[i].dtype,tab[i].type,tab[i].size);
}
}
static int row=1,col=1;
char buf[2048];
char dbuf[128];
int ind=0;
const char specialsymbols[]={'?',';',':',','};
const char arithmeticsymbols[]={'*'};
int charIs(int c,const char *arr)
{
int len;
if(arr==specialsymbols)
{
len=sizeof(specialsymbols)/sizeof(char);
}
else if(arr==arithmeticsymbols)
{
len=sizeof(arithmeticsymbols)/sizeof(char);
}
for(int i=0;i<len;i++)
{
if(c==arr[i])
{
return 1;
}
}
return 0;
}
void fillToken(struct token *tkn,char c,int row,int col, char *type)
{
tkn->row=row;
tkn->col=col;
strcpy(tkn->type,type);
tkn->lexeme[0]=c;
tkn->lexeme[1]='\0';
}
```

```
void newLine()
{
++row;
col=1;
}
int sz(char *w){
if(strcmp(w,"int")==0)
return 4;
if(strcmp(w,"char")==0)
return 1;
if(strcmp(w,"void")==0)
return 0;
if(strcmp(w,"float")==0)
return 8;
if(strcmp(w,"bool")==0)
return 1;
}
struct token getNextToken(FILE *fa)
{
int c;
struct token tkn=
{
.row=-1
};
int gotToken=0;
while(!gotToken && (c=fgetc(fa))!=EOF)
{
if(charIs(c,specialsymbols))
{
fillToken(&tkn,c,row,col,"SS");
gotToken=1;
++col;
}
else if(charIs(c,arithmeticsymbols))
{
fseek(fa,-1,SEEK_CUR);
c=getc(fa);
if(isalnum(c)){
fillToken(&tkn,c,row,col,"ARITHMETICOPERATOR");
gotToken=1;
++col;
}
fseek(fa,1,SEEK_CUR);
}
else if(c=='(')
```

```
{
fillToken(&tkn,c,row,col,"LB");
gotToken=1;
col++;
}
else if(c==')')
{
fillToken(&tkn,c,row,col,"RB");
gotToken=1;
col++;
}
else if(c=='{'){
fillToken(&tkn,c,row,col,"LC");
gotToken=1;
col++;
}
else if(c=='}')
{
fillToken(&tkn,c,row,col,"RC");
gotToken=1;
col++;
}
else if(c=='[')
{
fillToken(&tkn,c,row,col,"LS");
gotToken=1;
col++;
}
else if(c==']')
{
fillToken(&tkn,c,row,col,"RS");
gotToken=1;
col++;
}
else if(c=='+')
{
int x=fgetc(fa);
if(x!='+')
{
fillToken(&tkn,c,row,col,"ARITHMETICOPERATOR");
gotToken=1;
col++;
fseek(fa,-1,SEEK_CUR);
}
else
```

```
{
fillToken(&tkn,c,row,col,"UNARYOPERATOR");
strcpy(tkn.lexeme,"++");
gotToken=1;
col+=2;
}
}
else if(c=='-')
{
int x=fgetc(fa);
if(x!='-')
{
fillToken(&tkn,c,row,col,"ARITHMETICOPERATOR");
gotToken=1;
col++;
fseek(fa,-1,SEEK_CUR);
}
else{
fillToken(&tkn,c,row,col,"UNARYOPERATOR");
strcpy(tkn.lexeme,"++");
gotToken=1;
col+=2;
}
}
else if(c=='=')
{
int x=fgetc(fa);
if(x!='=')
{
fillToken(&tkn,c,row,col,"ASSIGNMENTOPERATOR");
gotToken=1;
col++;
fseek(fa,-1,SEEK_CUR);
}
else
{
fillToken(&tkn,c,row,col,"RELATIONALOPERATOR");
strcpy(tkn.lexeme,"++");
gotToken=1;
col+=2;
}
}
else if(isdigit(c))
{
fillToken(&tkn,c,row,col++,"NUMBER");
```

```
int j=1;
while((c=fgetc(fa))!=EOF && isdigit(c))
{
tkn.lexeme[j++]=c;
col++;
}
tkn.lexeme[j]='\0';
gotToken=1;
fseek(fa,-1,SEEK_CUR);
}
else if(c == '#')
{
while((c = fgetc(fa))!= EOF && c != '\n');
newLine();
}
else if(c=='\n')
{
newLine();
c = fgetc(fa);
if(c == '#')
{
while((c = fgetc(fa)) != EOF && c != '\n');
newLine();
}else if(c != EOF)
{
fseek(fa, -1, SEEK_CUR);
}
}
else if(isspace(c))
{
++col;
}
else if(isalpha(c) || c=='_')
{
tkn.row=row;
tkn.col=col++;
tkn.lexeme[0]=c;
int j=1;
while((c=fgetc(fa))!=EOF && isalnum(c))
{
tkn.lexeme[j++]=c;
col++;
}
tkn.lexeme[j]='\0';
if(isKeyword(tkn.lexeme))
```

```
{
strcpy(tkn.type,"KEYWORD");
}
else
{
strcpy(tkn.type,"IDENTIFIER");
}
gotToken=1;
fseek(fa,-1,SEEK_CUR);
}
else if(c=='/')
{
int d=fgetc(fa);
++col;
if(d=='/')
{
while((c=fgetc(fa))!= EOF && c!='\n')
{
++col;
}
if(c=='\n')
{
newLine();
}
}
else if(d=='*')
{
do
{
if(d=='\n'){
newLine();
}
while((c==fgetc(fa))!= EOF && c!='*')
{
++col;
if(c=='\n')
{
newLine();
}
}
++col;
}while((d==fgetc(fa))!= EOF && d!='/' && (++col));
++col;
}
else
```

```
{
fillToken(&tkn,c,row,--col,"ARITHMETIC OPERATOR");
gotToken=1;
fseek(fa,-1,SEEK_CUR);
}
}
else if(c=="")
{
tkn.row=row;
tkn.col=col;
strcpy(tkn.type, "STRING LITERAL");
int k = 1;
tkn.lexeme[0] = "";
while((c = fgetc(fa)) != EOF && c != "")
{
tkn.lexeme[k++] = c;
++col;
}
tkn.lexeme[k] = "";
gotToken = 1;
}
else if(c == '<' || c == '>' || c == '!')
{
fillToken(&tkn, c, row, col, "RELATIONALOPERATOR");
++col;
int d = fgetc(fa);
if(d == '=')
{
++col;
strcat(tkn.lexeme, "=");
}
else
{
if(c == '!')
{
strcpy(tkn.type, "LOGICALOPERATOR");}
fseek(fa, -1, SEEK_CUR);
}
gotToken = 1;
}
else if(c == '&' || c == '|')
{
int d = fgetc(fa);
if(c == d)
{
```

```
tkn.lexeme[0] = tkn.lexeme[1] = c;
tkn.lexeme[2] = '\0';
tkn.row = row;
tkn.col = col;
++col;
gotToken = 1;
strcpy(tkn.type, "LOGICALOPERATOR");
}
else
{
fseek(fa, -1, SEEK_CUR);
}
++col;
}
else
{
++col;
}
}
return tkn;
}
int main()
{
FILE *fa, *fb;
int ca, cb;
fa = fopen("input.c", "r");
if (fa == NULL){
printf("Cannot open file \n");
exit(0);
}
fb = fopen("out4.c", "w+");
ca = getc(fa);
while (ca != EOF){
if(ca==' ')
{
putc(ca,fb);
while(ca==' ')
ca = getc(fa);
}if (ca=='/')
{
cb = getc(fa);
if (cb == '/')
{
while(ca != '\n')
ca = getc(fa);
```

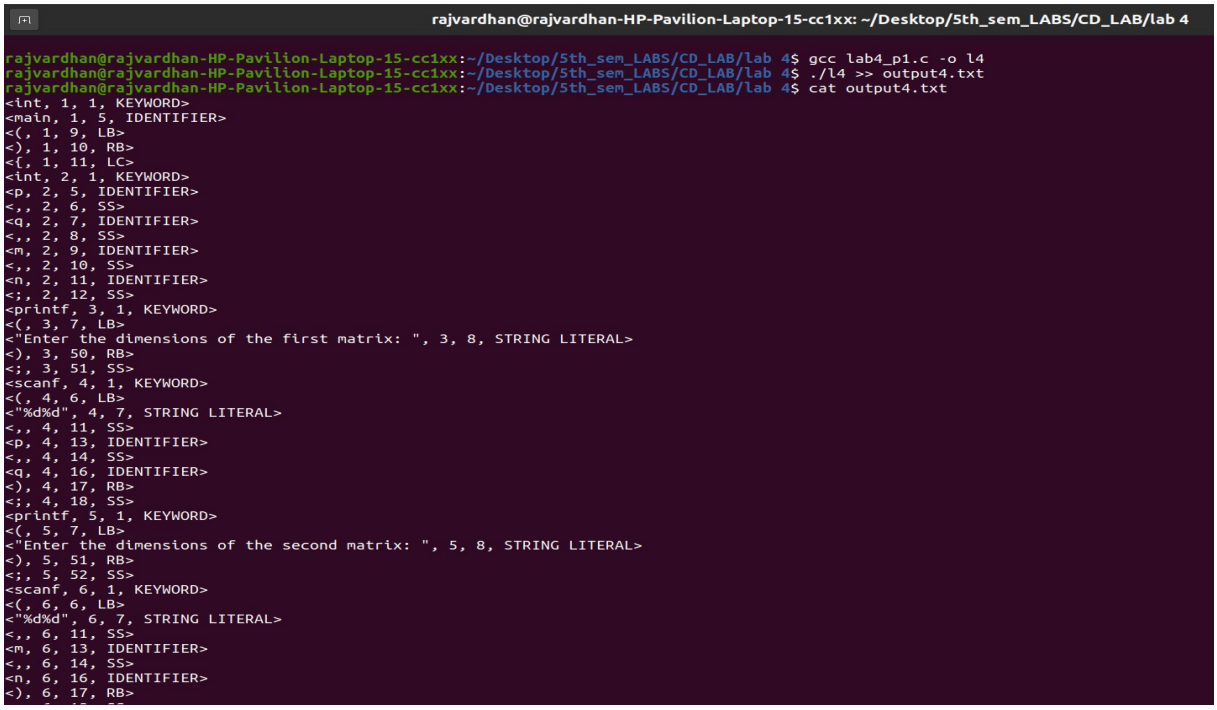
```
}
else if (cb == '*')
{
do
{
while(ca != '*')
ca = getc(fa);
ca = getc(fa);
} while (ca != '/');
}
else{
putc(ca,fb);
putc(cb,fb);
}
}
else putc(ca,fb);
ca = getc(fa);
}
fclose(fa);
fclose(fb);
fa = fopen("out4.c", "r");
if(fa == NULL){
printf("Cannot open file");
return 0;
}
fb = fopen("temp.c", "w+");
ca = getc(fa);
while (ca != EOF)
{
if(ca=='"')
{
putc(ca,fb);
ca=getc(fa);
while(ca!='"')
{
putc(ca,fb);
ca=getc(fa);
}
}
else if(ca=='#')
{
while(ca!='\n')
{ca=getc(fa);
}
ca=getc(fa);
```

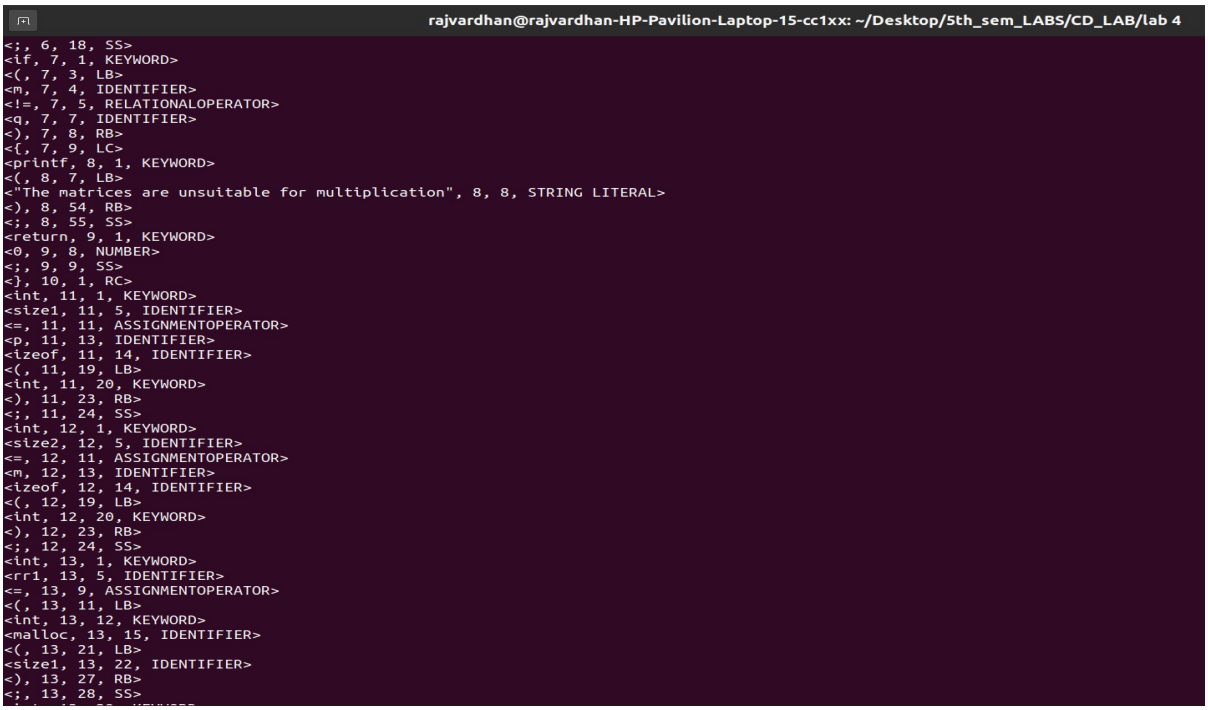
```
}
putc(ca,fb);
ca = getc(fa);
}
fclose(fa);
fclose(fb);
fa = fopen("temp.c", "r");
fb = fopen("out4.c", "w");
ca = getc(fa);
while(ca != EOF){
putc(ca, fb);
ca = getc(fa);
}
fclose(fa);
fclose(fb);
remove("temp.c");
FILE *f1=fopen("out4.c","r");
if(f1==NULL)
{
printf("Error! File cannot be opened!\n");
return 0;
}
struct token tkn;
struct sttable st[10][100];
int flag=0,i=0,j=0;
int tabsz[10];
char w[25];
w[0]='\0';
while((tkn=getNextToken(f1)).row!=-1)
{
printf("<%s, %d, %d, %s>\n",tkn.lexeme,tkn.row,tkn.col,tkn.type);
if(strcmp(tkn.type,"KEYWORD")==0)
{
if(isdtype(tkn.lexeme)==1)
{
strcpy(dbuf,tkn.lexeme);
}
}
else if(strcmp(tkn.type,"IDENTIFIER")==0)
{
strcpy(w,tkn.lexeme);
tkn=getNextToken(f1);
printf("<%s, %d, %d, %s>\n",tkn.lexeme,tkn.row,tkn.col,tkn.type);
if((strcmp(tkn.type,"LB"))==0)
{if(findTable(st[i],w,j)==0)
```

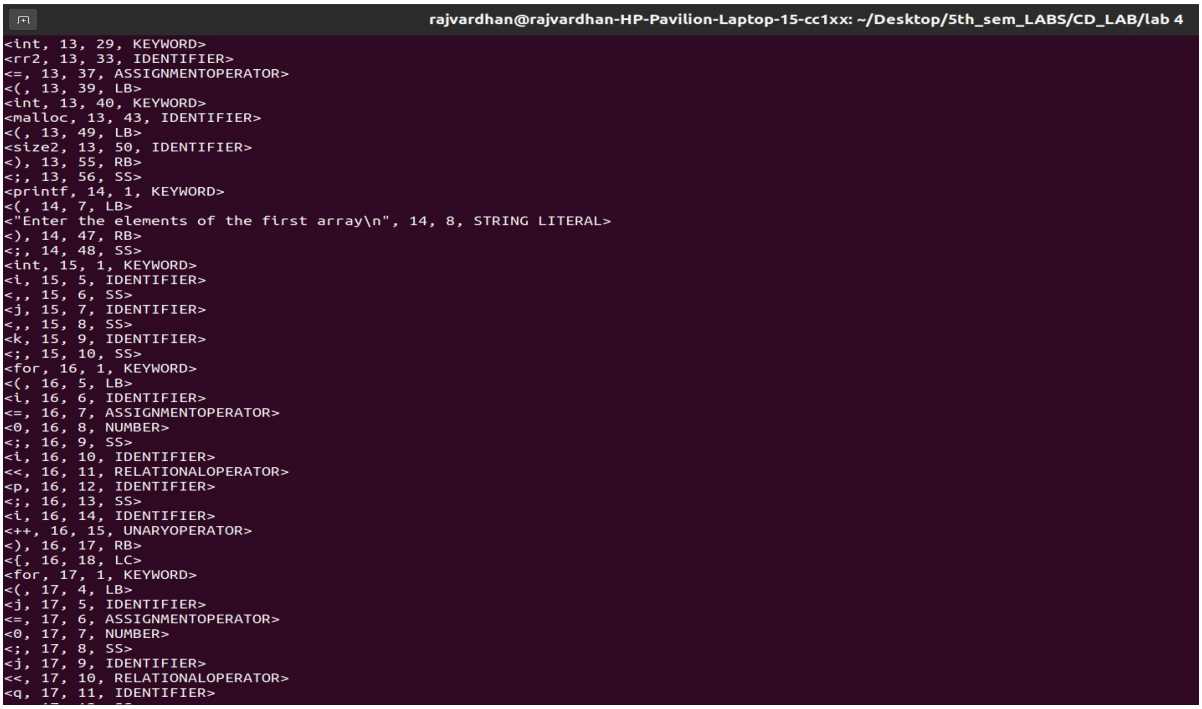
```
{
ind++;
st[i][j++]=fillTable(ind,w,dbuf,"func",-1);
}}
if((strcmp(tkn.type,"LS"))==0)
{
if(findTable(st[i],w,j)==0)
{
tkn=getNextToken(f1);
printf("<%s, %d, %d, %s>\n",tkn.lexeme,tkn.row,tkn.col,tkn.type);
int s=0;
if(strcmp(tkn.type,"NUMBER")==0)
{
s=atoi(tkn.lexeme);
}
ind++;
st[i][j++]=fillTable(ind,w,dbuf,"id",sz(dbuf)*s);
}}
else
{
if(findTable(st[i],w,j)==0)
{
ind++;
st[i][j++]=fillTable(ind,w,dbuf,"id",sz(dbuf));
}}}
else if(strcmp(tkn.type,"LC")==0)
{
flag++;
}
else if(strcmp(tkn.type,"RC")==0)
{
flag--;
if(flag==0)
{
tabsz[i]=j;
i++;
j=0;
ind=0;
}}}
int k=0;
printf("\n\n\nSYMBOL TABLEs STARTS
HERE---------------------------------------------->\n\n");
for(k=0;k<i;k++){
printTable(st[k],tabsz[k]);
printf("_____\n\n");
```

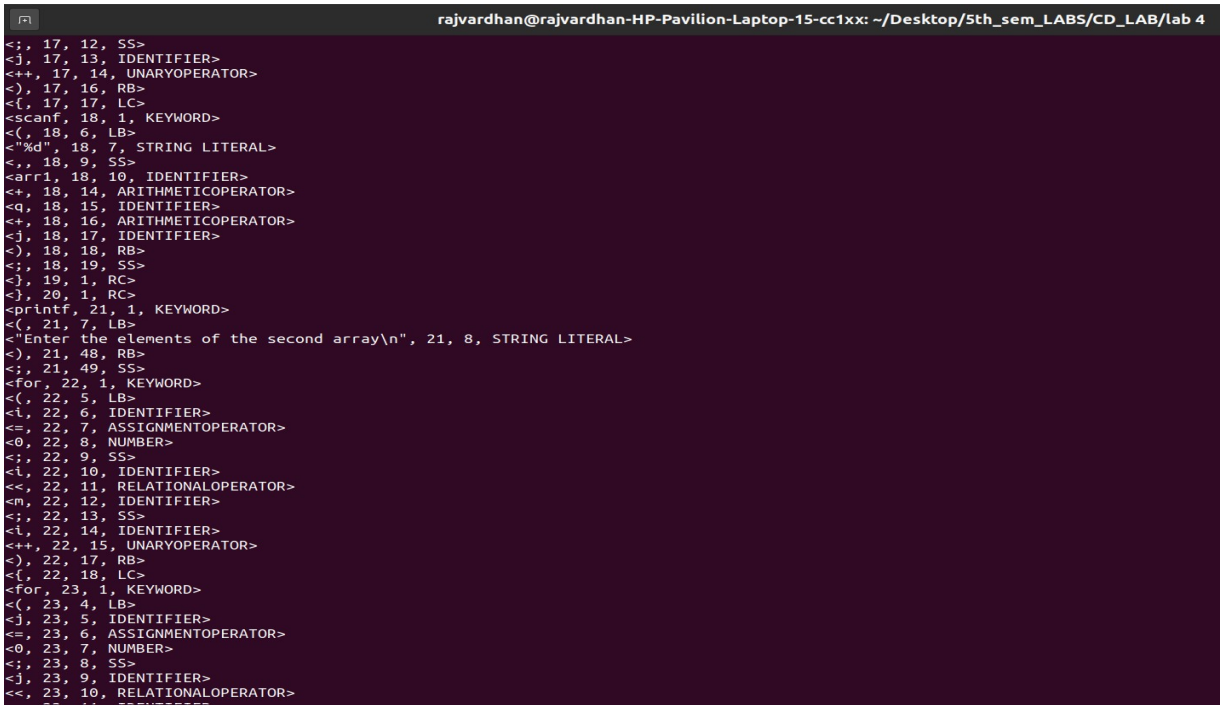
```
}
fclose(f1);
}
```

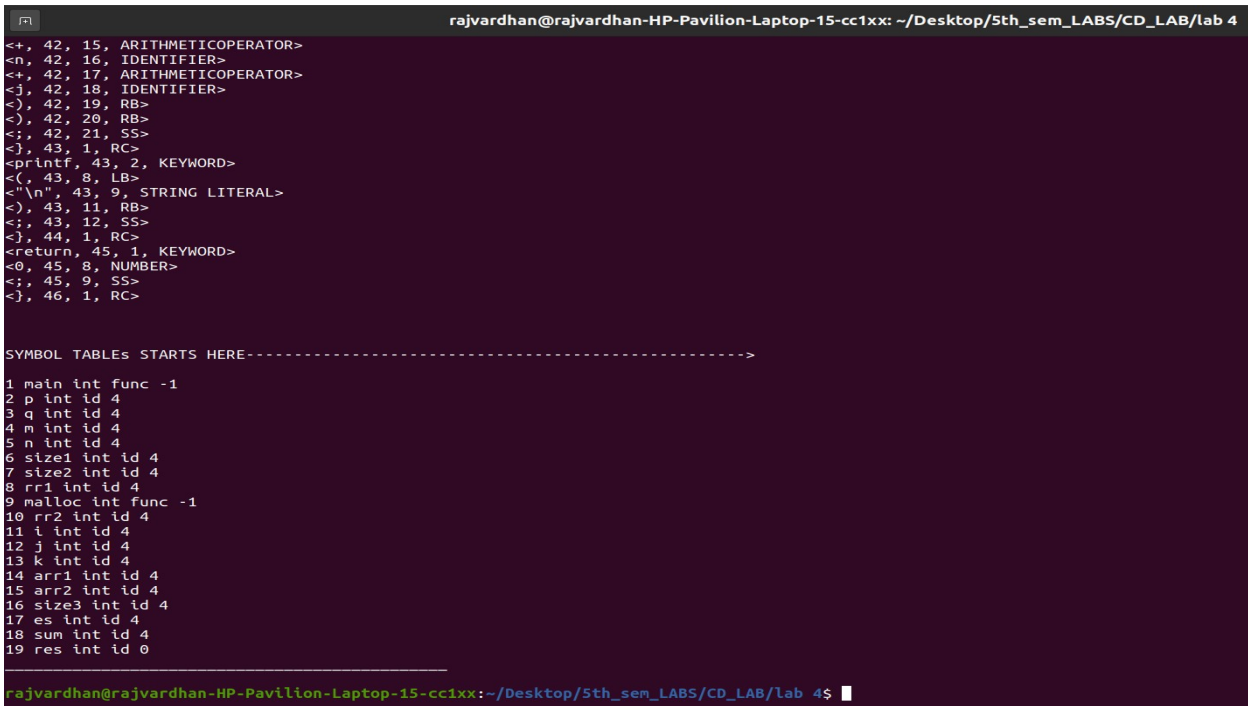
Output : (Output on Terminal)






rajvardhan@rajvardhan-HP-Pavilion-Laptop-15-cc1xx: ~/Desktop/5th_sem_LABS/CD_LAB/lab 4
<int, 13, 29, KEYWORD>
<rr2, 13, 33, IDENTIFIER>
<=, 13, 37, ASSIGNMENTOPERATOR>
<(, 13, 39, LB>
<int, 13, 40, KEYWORD>
<malloc, 13, 43, IDENTIFIER>
<(, 13, 49, LB>
<size2, 13, 50, IDENTIFIER>
<), 13, 55, RB>
<;, 13, 56, SS>
<printf, 14, 1, KEYWORD>
<(, 14, 7, LB>
<"Enter the elements of the first array\n", 14, 8, STRING LITERAL>
<), 14, 47, RB>
<;, 14, 48, SS>
<int, 15, 1, KEYWORD>
<i, 15, 5, IDENTIFIER>
<,, 15, 6, SS>
<j, 15, 7, IDENTIFIER>
<,, 15, 8, SS>
<k, 15, 9, IDENTIFIER>
<;, 15, 10, SS>
<for, 16, 1, KEYWORD>
<(, 16, 5, LB>
<l, 16, 6, IDENTIFIER>
<=, 16, 7, ASSIGNMENTOPERATOR>
<0, 16, 8, NUMBER>
<;, 16, 9, SS>
<i, 16, 10, IDENTIFIER>
<<, 16, 11, RELATIONALOPERATOR>
<p, 16, 12, IDENTIFIER>
<;, 16, 13, SS>
<i, 16, 14, IDENTIFIER>
<++, 16, 15, UNARYOPERATOR>
<), 16, 17, RB>
<{, 16, 18, LC>
<for, 17, 1, KEYWORD>
<(, 17, 4, LB>
<j, 17, 5, IDENTIFIER>
<=, 17, 6, ASSIGNMENTOPERATOR>
<0, 17, 7, NUMBER>
<;, 17, 8, SS>
<j, 17, 9, IDENTIFIER>
<<, 17, 10, RELATIONALOPERATOR>
<q, 17, 11, IDENTIFIER>


rajvardhan@rajvardhan-HP-Pavilion-Laptop-15-cc1xx: ~/Desktop/5th_sem_LABS/CD_LAB/lab 4
<;, 17, 12, SS>
<j, 17, 13, IDENTIFIER>
<++, 17, 14, UNARYOPERATOR>
<), 17, 16, RB>
<{, 17, 17, LC>
<scanf, 18, 1, KEYWORD>
<(, 18, 6, LB>
<"%d", 18, 7, STRING LITERAL>
<,, 18, 9, SS>
<arr1, 18, 10, IDENTIFIER>
<+, 18, 14, ARITHMETICOPERATOR>
<q, 18, 15, IDENTIFIER>
<+, 18, 16, ARITHMETICOPERATOR>
<j, 18, 17, IDENTIFIER>
<), 18, 18, RB>
<;, 18, 19, SS>
<}, 19, 1, RC>
<}, 20, 1, RC>
<printf, 21, 1, KEYWORD>
<(, 21, 7, LB>
<"Enter the elements of the second array\n", 21, 8, STRING LITERAL>
<), 21, 48, RB>
<;, 21, 49, SS>
<for, 22, 1, KEYWORD>
<(, 22, 5, LB>
<i, 22, 6, IDENTIFIER>
<=, 22, 7, ASSIGNMENTOPERATOR>
<0, 22, 8, NUMBER>
<;, 22, 9, SS>
<i, 22, 10, IDENTIFIER>
<<, 22, 11, RELATIONALOPERATOR>
<m, 22, 12, IDENTIFIER>
<;, 22, 13, SS>
<i, 22, 14, IDENTIFIER>
<++, 22, 15, UNARYOPERATOR>
<), 22, 17, RB>
<{, 22, 18, LC>
<for, 23, 1, KEYWORD>
<(, 23, 4, LB>
<j, 23, 5, IDENTIFIER>
<=, 23, 6, ASSIGNMENTOPERATOR>
<0, 23, 7, NUMBER>
<;, 23, 8, SS>
<j, 23, 9, IDENTIFIER>
<<, 23, 10, RELATIONALOPERATOR>



out4.c :

```
int main(){
int p,q,m,n;
printf("Enter the dimensions of the first matrix: ");
scanf("%d%d",&p,&q);
printf("Enter the dimensions of the second matrix: ");
scanf("%d%d",&m,&n);
if(m!=q){
printf("The matrices are unsuitable for multiplication");
return 0;
}
int size1 = p*q*sizeof(int);
int size2 = m*n*sizeof(int);
int *arr1 = (int*)malloc(size1);int *arr2 = (int*)malloc(size2);
printf("Enter the elements of the first array\n");
int i,j,k;
for (i=0;i<p;i++){
for(j=0;j<q;j++){
scanf("%d",arr1+q*i+j);
}
}
printf("Enter the elements of the second array\n");
for (i=0;i<m;i++){
for(j=0;j<n;j++){
```

```
scanf("%d",arr2+n*i+j);
}
}
int size3 = p*n*sizeof(int);
int *res = (int*)malloc(size3);
int sum = 0;
for(i=0;i<p;i++){
for(j=0;j<n;j++){
for(k=0;k<m;k++){
sum+= arr1[i*m+k] * arr2[k*n+j];
}
res[i*n+j] = sum;
sum=0;
}
}
printf("The result is\n");
for(i=0;i<p;i++){
for(j=0;j<n;j++){
printf("%d ",*(res+n*i+j));
}printf("\n");
}
return 0;
}
```

output4.txt: (Output in a File)

```
<int, 1, 1, KEYWORD>
<main, 1, 5, IDENTIFIER>
<(, 1, 9, LB>
<), 1, 10, RB>
<{, 1, 11, LC>
<int, 2, 1, KEYWORD>
<p, 2, 5, IDENTIFIER>
<,, 2, 6, SS>
<q, 2, 7, IDENTIFIER>
<,, 2, 8, SS>
<m, 2, 9, IDENTIFIER>
<,, 2, 10, SS>
<n, 2, 11, IDENTIFIER>
<;, 2, 12, SS>
<printf, 3, 1, KEYWORD>
<(, 3, 7, LB>
<"Enter the dimensions of the first matrix: ", 3, 8, STRING LITERAL>
<), 3, 50, RB>
```

<;, 3, 51, SS>
<scanf, 4, 1, KEYWORD>
<(, 4, 6, LB>
<"%d%d", 4, 7, STRING LITERAL>
<,, 4, 11, SS>
<p, 4, 13, IDENTIFIER>
<,, 4, 14, SS>
<q, 4, 16, IDENTIFIER>
<), 4, 17, RB>
<;, 4, 18, SS>
<printf, 5, 1, KEYWORD>
<(, 5, 7, LB>
<"Enter the dimensions of the second matrix: ", 5, 8, STRING LITERAL>
<), 5, 51, RB>
<;, 5, 52, SS>
<scanf, 6, 1, KEYWORD>
<(, 6, 6, LB>
<"%d%d", 6, 7, STRING LITERAL>
<,, 6, 11, SS>
<m, 6, 13, IDENTIFIER>
<,, 6, 14, SS>
<n, 6, 16, IDENTIFIER>
<), 6, 17, RB>
<;, 6, 18, SS>
<if, 7, 1, KEYWORD>
<(, 7, 3, LB>
<m, 7, 4, IDENTIFIER>
<!=, 7, 5, RELATIONALOPERATOR>
<q, 7, 7, IDENTIFIER>
<), 7, 8, RB>
<{, 7, 9, LC>
<printf, 8, 1, KEYWORD>
<(, 8, 7, LB>
<"The matrices are unsuitable for multiplication", 8, 8, STRING LITERAL>
<), 8, 54, RB>
<;, 8, 55, SS>
<return, 9, 1, KEYWORD>
<0, 9, 8, NUMBER>
<;, 9, 9, SS>
<}, 10, 1, RC>
<int, 11, 1, KEYWORD>
<size1, 11, 5, IDENTIFIER>
<=, 11, 11, ASSIGNMENTOPERATOR>
<p, 11, 13, IDENTIFIER>
<izeof, 11, 14, IDENTIFIER>

<(, 11, 19, LB>
<int, 11, 20, KEYWORD>
<), 11, 23, RB>
<;, 11, 24, SS>
<int, 12, 1, KEYWORD>
<size2, 12, 5, IDENTIFIER>
<=, 12, 11, ASSIGNMENTOPERATOR>
<m, 12, 13, IDENTIFIER>
<izeof, 12, 14, IDENTIFIER>
<(, 12, 19, LB>
<int, 12, 20, KEYWORD>
<), 12, 23, RB>
<;, 12, 24, SS>
<int, 13, 1, KEYWORD>
<rr1, 13, 5, IDENTIFIER>
<=, 13, 9, ASSIGNMENTOPERATOR>
<(, 13, 11, LB>
<int, 13, 12, KEYWORD>
<malloc, 13, 15, IDENTIFIER>
<(, 13, 21, LB>
<size1, 13, 22, IDENTIFIER>
<), 13, 27, RB>
<;, 13, 28, SS>
<int, 13, 29, KEYWORD>
<rr2, 13, 33, IDENTIFIER>
<=, 13, 37, ASSIGNMENTOPERATOR>
<(, 13, 39, LB>
<int, 13, 40, KEYWORD>
<malloc, 13, 43, IDENTIFIER>
<(, 13, 49, LB>
<size2, 13, 50, IDENTIFIER>
<), 13, 55, RB>
<;, 13, 56, SS>
<printf, 14, 1, KEYWORD>
<(, 14, 7, LB>
<"Enter the elements of the first array\n", 14, 8, STRING LITERAL>
<), 14, 47, RB>
<;, 14, 48, SS>
<int, 15, 1, KEYWORD>
<i, 15, 5, IDENTIFIER>
<,, 15, 6, SS>
<j, 15, 7, IDENTIFIER>
<,, 15, 8, SS>
<k, 15, 9, IDENTIFIER>
<;, 15, 10, SS>

<for, 16, 1, KEYWORD>
<(, 16, 5, LB>
<i, 16, 6, IDENTIFIER>
<=, 16, 7, ASSIGNMENTOPERATOR>
<0, 16, 8, NUMBER>
<;, 16, 9, SS>
<i, 16, 10, IDENTIFIER>
<<, 16, 11, RELATIONALOPERATOR>
<p, 16, 12, IDENTIFIER>
<;, 16, 13, SS>
<i, 16, 14, IDENTIFIER>
<++, 16, 15, UNARYOPERATOR>
<), 16, 17, RB>
<{, 16, 18, LC>
<for, 17, 1, KEYWORD>
<(, 17, 4, LB>
<j, 17, 5, IDENTIFIER>
<=, 17, 6, ASSIGNMENTOPERATOR>
<0, 17, 7, NUMBER>
<;, 17, 8, SS>
<j, 17, 9, IDENTIFIER>
<<, 17, 10, RELATIONALOPERATOR>
<q, 17, 11, IDENTIFIER>
<;, 17, 12, SS>
<j, 17, 13, IDENTIFIER>
<++, 17, 14, UNARYOPERATOR>
<), 17, 16, RB>
<{, 17, 17, LC>
<scanf, 18, 1, KEYWORD>
<(, 18, 6, LB>
<"%d", 18, 7, STRING LITERAL>
<,, 18, 9, SS>
<arr1, 18, 10, IDENTIFIER>
<+, 18, 14, ARITHMETICOPERATOR>
<q, 18, 15, IDENTIFIER>
<+, 18, 16, ARITHMETICOPERATOR>
<j, 18, 17, IDENTIFIER>
<), 18, 18, RB>
<;, 18, 19, SS>
<}, 19, 1, RC>
<}, 20, 1, RC>
<printf, 21, 1, KEYWORD>
<(, 21, 7, LB>
<"Enter the elements of the second array\n", 21, 8, STRING LITERAL>
<), 21, 48, RB>

<;, 21, 49, SS>
<for, 22, 1, KEYWORD>
<(, 22, 5, LB>
<i, 22, 6, IDENTIFIER>
<=, 22, 7, ASSIGNMENTOPERATOR>
<0, 22, 8, NUMBER>
<;, 22, 9, SS>
<i, 22, 10, IDENTIFIER>
<<, 22, 11, RELATIONALOPERATOR>
<m, 22, 12, IDENTIFIER>
<;, 22, 13, SS>
<i, 22, 14, IDENTIFIER>
<++, 22, 15, UNARYOPERATOR>
<), 22, 17, RB>
<{, 22, 18, LC>
<for, 23, 1, KEYWORD>
<(, 23, 4, LB>
<j, 23, 5, IDENTIFIER>
<=, 23, 6, ASSIGNMENTOPERATOR>
<0, 23, 7, NUMBER>
<;, 23, 8, SS>
<j, 23, 9, IDENTIFIER>
<<, 23, 10, RELATIONALOPERATOR>
<n, 23, 11, IDENTIFIER>
<;, 23, 12, SS>
<j, 23, 13, IDENTIFIER>
<++, 23, 14, UNARYOPERATOR>
<), 23, 16, RB>
<{, 23, 17, LC>
<scanf, 24, 1, KEYWORD>
<(, 24, 6, LB>
<"%d", 24, 7, STRING LITERAL>
<,, 24, 9, SS>
<arr2, 24, 10, IDENTIFIER>
<+, 24, 14, ARITHMETICOPERATOR>
<n, 24, 15, IDENTIFIER>
<+, 24, 16, ARITHMETICOPERATOR>
<j, 24, 17, IDENTIFIER>
<), 24, 18, RB>
<;, 24, 19, SS>
<}, 25, 1, RC>
<}, 26, 1, RC>
<int, 27, 1, KEYWORD>
<size3, 27, 5, IDENTIFIER>
<=, 27, 11, ASSIGNMENTOPERATOR>

<p, 27, 13, IDENTIFIER>
<izeof, 27, 14, IDENTIFIER>
<(, 27, 19, LB>
<int, 27, 20, KEYWORD>
<), 27, 23, RB>
<;, 27, 24, SS>
<int, 28, 1, KEYWORD>
<es, 28, 5, IDENTIFIER>
<=, 28, 8, ASSIGNMENTOPERATOR>
<(, 28, 10, LB>
<int, 28, 11, KEYWORD>
<malloc, 28, 14, IDENTIFIER>
<(, 28, 20, LB>
<size3, 28, 21, IDENTIFIER>
<), 28, 26, RB>
<;, 28, 27, SS>
<int, 29, 1, KEYWORD>
<sum, 29, 5, IDENTIFIER>
<=, 29, 9, ASSIGNMENTOPERATOR>
<0, 29, 11, NUMBER>
<;, 29, 12, SS>
<for, 30, 1, KEYWORD>
<(, 30, 4, LB>
<i, 30, 5, IDENTIFIER>
<=, 30, 6, ASSIGNMENTOPERATOR>
<0, 30, 7, NUMBER>
<;, 30, 8, SS>
<i, 30, 9, IDENTIFIER>
<<, 30, 10, RELATIONALOPERATOR>
<p, 30, 11, IDENTIFIER>
<;, 30, 12, SS>
<i, 30, 13, IDENTIFIER>
<++, 30, 14, UNARYOPERATOR>
<), 30, 16, RB>
<{, 30, 17, LC>
<for, 31, 1, KEYWORD>
<(, 31, 4, LB>
<j, 31, 5, IDENTIFIER>
<=, 31, 6, ASSIGNMENTOPERATOR>
<0, 31, 7, NUMBER>
<;, 31, 8, SS>
<j, 31, 9, IDENTIFIER>
<<, 31, 10, RELATIONALOPERATOR>
<n, 31, 11, IDENTIFIER>
<;, 31, 12, SS>

<j, 31, 13, IDENTIFIER>
<++, 31, 14, UNARYOPERATOR>
<), 31, 16, RB>
<{, 31, 17, LC>
<for, 32, 1, KEYWORD>
<(, 32, 4, LB>
<k, 32, 5, IDENTIFIER>
<=, 32, 6, ASSIGNMENTOPERATOR>
<0, 32, 7, NUMBER>
<;, 32, 8, SS>
<k, 32, 9, IDENTIFIER>
<<, 32, 10, RELATIONALOPERATOR>
<m, 32, 11, IDENTIFIER>
<;, 32, 12, SS>
<k, 32, 13, IDENTIFIER>
<++, 32, 14, UNARYOPERATOR>
<), 32, 16, RB>
<{, 32, 17, LC>
<sum, 33, 1, IDENTIFIER>
<+, 33, 4, ARITHMETICOPERATOR>
<=, 33, 5, ASSIGNMENTOPERATOR>
<arr1, 33, 7, IDENTIFIER>
<[, 33, 11, LS>
<i, 33, 12, IDENTIFIER>
<+, 33, 13, ARITHMETICOPERATOR>
<k, 33, 14, IDENTIFIER>
<], 33, 15, RS>
<arr2, 33, 17, IDENTIFIER>
<[, 33, 21, LS>
<k, 33, 22, IDENTIFIER>
<+, 33, 23, ARITHMETICOPERATOR>
<j, 33, 24, IDENTIFIER>
<], 33, 25, RS>
<;, 33, 26, SS>
<}, 34, 1, RC>
<res, 35, 1, IDENTIFIER>
<[, 35, 4, LS>
<i, 35, 5, IDENTIFIER>
<+, 35, 6, ARITHMETICOPERATOR>
<j, 35, 7, IDENTIFIER>
<], 35, 8, RS>
<=, 35, 10, ASSIGNMENTOPERATOR>
<sum, 35, 12, IDENTIFIER>
<;, 35, 15, SS>
<sum, 36, 1, IDENTIFIER>

<=, 36, 4, ASSIGNMENTOPERATOR>
<0, 36, 5, NUMBER>
<;, 36, 6, SS>
<}, 37, 1, RC>
<}, 38, 1, RC>
<printf, 39, 1, KEYWORD>
<(, 39, 7, LB>
<"The result is\n", 39, 8, STRING LITERAL>
<), 39, 23, RB>
<;, 39, 24, SS>
<for, 40, 1, KEYWORD>
<(, 40, 4, LB>
<i, 40, 5, IDENTIFIER>
<=, 40, 6, ASSIGNMENTOPERATOR>
<0, 40, 7, NUMBER>
<;, 40, 8, SS>
<i, 40, 9, IDENTIFIER>
<<, 40, 10, RELATIONALOPERATOR>
<p, 40, 11, IDENTIFIER>
<;, 40, 12, SS>
<i, 40, 13, IDENTIFIER>
<++, 40, 14, UNARYOPERATOR>
<), 40, 16, RB>
<{, 40, 17, LC>
<for, 41, 1, KEYWORD>
<(, 41, 4, LB>
<j, 41, 5, IDENTIFIER>
<=, 41, 6, ASSIGNMENTOPERATOR>
<0, 41, 7, NUMBER>
<;, 41, 8, SS>
<j, 41, 9, IDENTIFIER>
<<, 41, 10, RELATIONALOPERATOR>
<n, 41, 11, IDENTIFIER>
<;, 41, 12, SS>
<j, 41, 13, IDENTIFIER>
<++, 41, 14, UNARYOPERATOR>
<), 41, 16, RB>
<{, 41, 17, LC>
<printf, 42, 1, KEYWORD>
<(, 42, 7, LB>
<"%d ", 42, 8, STRING LITERAL>
<,, 42, 11, SS>
<res, 42, 12, IDENTIFIER>
<+, 42, 15, ARITHMETICOPERATOR>
<n, 42, 16, IDENTIFIER>

<+, 42, 17, ARITHMETICOPERATOR>
<j, 42, 18, IDENTIFIER>
<), 42, 19, RB>
<), 42, 20, RB>
<;, 42, 21, SS>
<}, 43, 1, RC>
<printf, 43, 2, KEYWORD>
<(, 43, 8, LB>
<"\n", 43, 9, STRING LITERAL>
<), 43, 11, RB>
<;, 43, 12, SS>
<}, 44, 1, RC>
<return, 45, 1, KEYWORD>
<0, 45, 8, NUMBER>
<;, 45, 9, SS>
<}, 46, 1, RC>

SYMBOL TABLEs STARTS HERE---------->

1 main int func -1
2 p int id 4
3 q int id 4
4 m int id 4
5 n int id 4
6 size1 int id 4
7 size2 int id 4
8 rr1 int id 4
9 malloc int func -1
10 rr2 int id 4
11 i int id 4
12 j int id 4
13 k int id 4
14 arr1 int id 4
15 arr2 int id 4
16 size3 int id 4
17 es int id 4
18 sum int id 4
19 res int id 0