LAB-4 Compiler Design -> Symbol Table

Name : Sagnik Chatterjee

Roll No :61

Sec :B

Reg: 180905478

Q1

Code:

/\*

AUTHOR : SAGNIK CHATTERJEE

DATE :11 DEC,2020

USAGE : ./q1 sampleIn.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 \*datatypes[]={"int","char","void","float","bool"};

int isdatatype(char \*word){

//to check if its datatype or not

int i;

for(i=0;i<sizeof(datatypes)/sizeof(char\*);i++){

if(strcmp(word,datatypes[i])==0){

return 1;

}

}

return 0;

}

int isKeyword(char \*word){

int i;

for(i=0;i<sizeof(keywords)/sizeof(char\*);i++){

if(strcmp(word,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 %d\n",tab[i].sno,tab[i].lexeme,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){

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);

gotToken=1;

++col;

}

else if(charIs(c,arithmeticsymbols))

{

fseek(fa,-1,SEEK\_CUR);

c=getc(fa);

if(isalnum(c)){

fillToken(&tkn,c,row,col);

gotToken=1;

++col;

}

fseek(fa,1,SEEK\_CUR);

}

else if(c=='(')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

}

else if(c==')')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

}

else if(c=='{')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

}

else if(c=='}')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

}

else if(c=='[')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

}

else if(c==']')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

}

else if(c=='+')

{

int x=fgetc(fa);

if(x!='+')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

fseek(fa,-1,SEEK\_CUR);

}

else

{

fillToken(&tkn,c,row,col);

strcpy(tkn.lexeme,"++");

gotToken=1;

col+=2;

}

}

else if(c=='-')

{

int x=fgetc(fa);

if(x!='-')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

fseek(fa,-1,SEEK\_CUR);

}

else

{

fillToken(&tkn,c,row,col);

strcpy(tkn.lexeme,"++");

gotToken=1;

col+=2;

}

}

else if(c=='=')

{

int x=fgetc(fa);

if(x!='=')

{

fillToken(&tkn,c,row,col);

gotToken=1;

col++;

fseek(fa,-1,SEEK\_CUR);

}

else

{

fillToken(&tkn,c,row,col);

strcpy(tkn.lexeme,"++");

gotToken=1;

col+=2;

}

}

else if(isdigit(c))

{

fillToken(&tkn,c,row,col++);

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);

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);

++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("sampleIn.c", "r");

if (fa == NULL){

printf("[ERROR] Cannot open file for reading . \n");

exit(0);

}

fb = fopen("sampleout.c", "w+");

if (fb == NULL){

printf("[ERROR] Cannot open file for writing. \n");

exit(0);

}

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("sampleout.c", "r");

if(fa == NULL){

printf("[ERROR] Cannot open file");

return 0;

}

fb = fopen("temp.c", "w+");

if (fb == NULL){

printf("[ERROR] Cannot open file for writing. \n");

exit(0);

}

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("sampleout.c", "w");

ca = getc(fa);

while(ca != EOF){

putc(ca, fb);

ca = getc(fa);

}

fclose(fa);

fclose(fb);

remove("temp.c");

FILE \*f1=fopen("sampleout.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>\n",tkn.lexeme,tkn.row,tkn.col);

if(strcmp(tkn.type,"KEYWORD")==0)

{

if(isdatatype(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>\n",tkn.lexeme,tkn.row,tkn.col);

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>\n",tkn.lexeme,tkn.row,tkn.col);

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;

for(k=0;k<i;k++)

{

printTable(st[k],tabsz[k]);

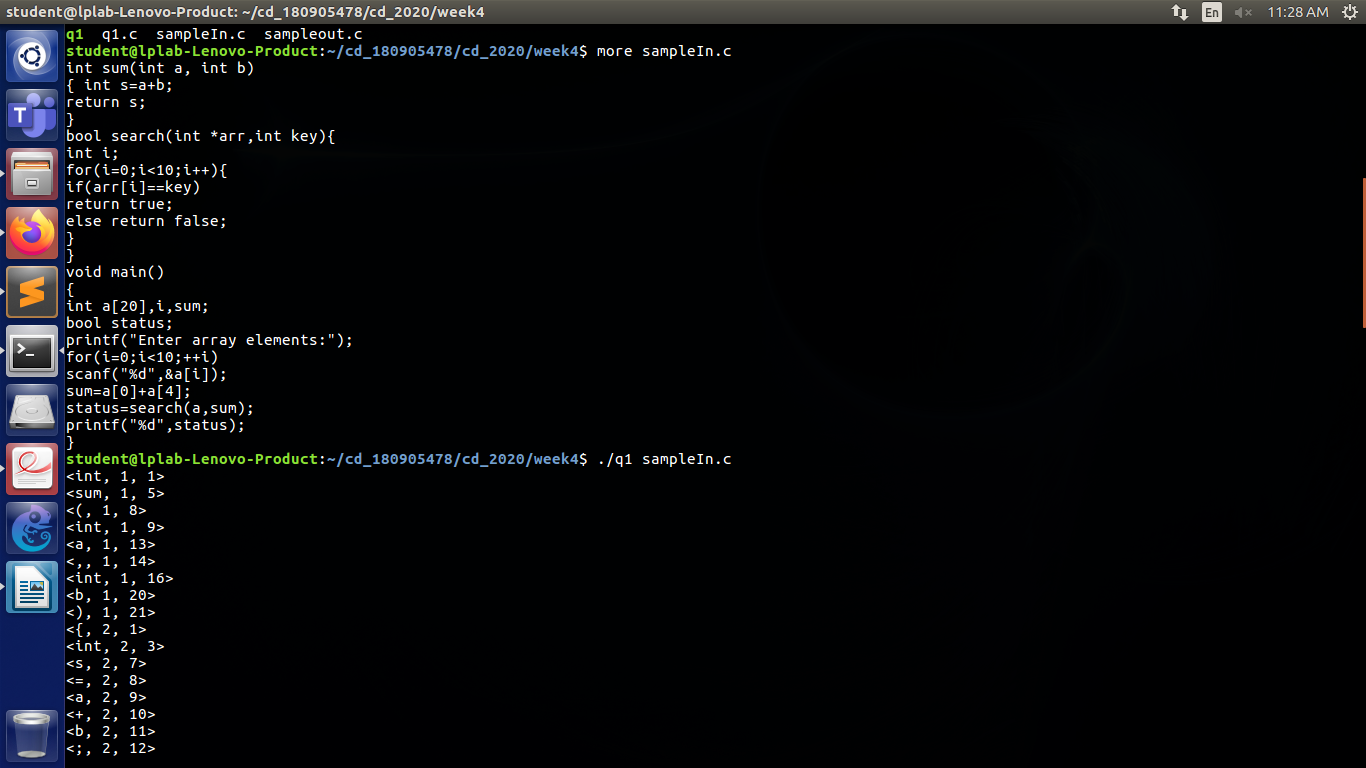
printf("\*\*\*\*\*\*\*\*-------\*\*\*\*\*\*\*\*\*------\*\*\*\*\*\*\*\*\n\n");

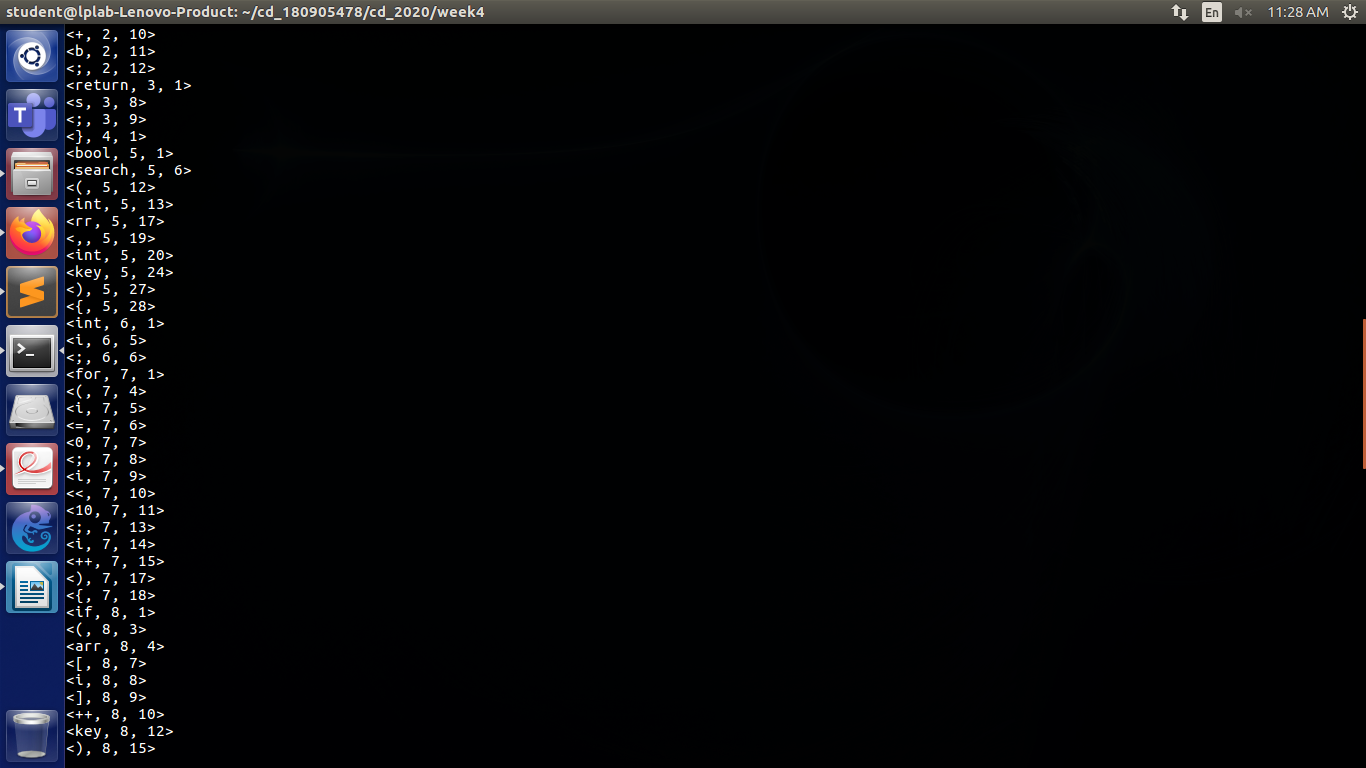
}

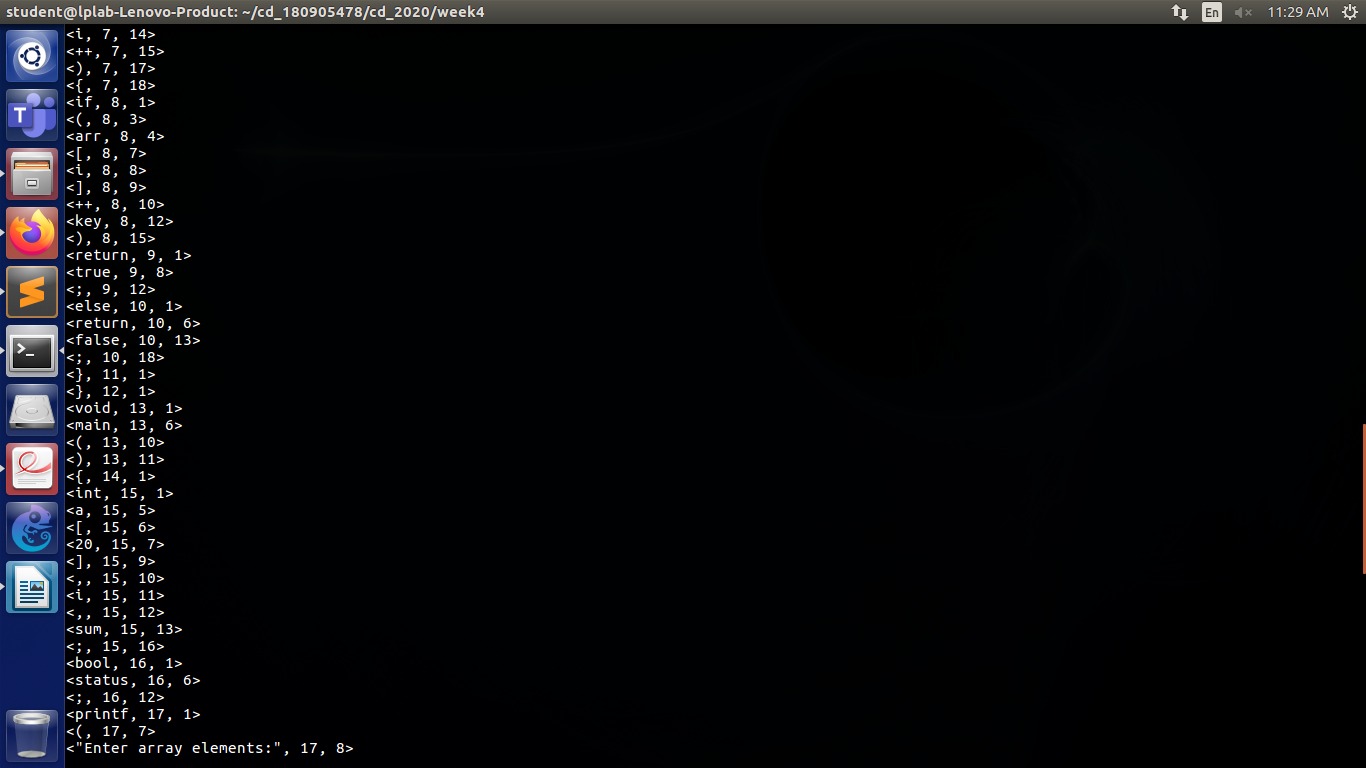
fclose(f1);

}

Screenshot:







sampleIn.c :

int sum(int a, int b)

{ int s=a+b;

return s;

}

bool search(int \*arr,int key){

int i;

for(i=0;i<10;i++){

if(arr[i]==key)

return true;

else return false;

}

}

void main()

{

int a[20],i,sum;

bool status;

printf("Enter array elements:");

for(i=0;i<10;++i)

scanf("%d",&a[i]);

sum=a[0]+a[4];

status=search(a,sum);

printf("%d",status);

}

The final symbol table constructed was:

<int, 1, 1>

<sum, 1, 5>

<(, 1, 8>

<int, 1, 9>

<a, 1, 13>

<,, 1, 14>

<int, 1, 16>

<b, 1, 20>

<), 1, 21>

<{, 2, 1>

<int, 2, 3>

<s, 2, 7>

<=, 2, 8>

<a, 2, 9>

<+, 2, 10>

<b, 2, 11>

<;, 2, 12>

<return, 3, 1>

<s, 3, 8>

<;, 3, 9>

<}, 4, 1>

<bool, 5, 1>

<search, 5, 6>

<(, 5, 12>

<int, 5, 13>

<rr, 5, 17>

<,, 5, 19>

<int, 5, 20>

<key, 5, 24>

<), 5, 27>

<{, 5, 28>

<int, 6, 1>

<i, 6, 5>

<;, 6, 6>

<for, 7, 1>

<(, 7, 4>

<i, 7, 5>

<=, 7, 6>

<0, 7, 7>

<;, 7, 8>

<i, 7, 9>

<<, 7, 10>

<10, 7, 11>

<;, 7, 13>

<i, 7, 14>

<++, 7, 15>

<), 7, 17>

<{, 7, 18>

<if, 8, 1>

<(, 8, 3>

<arr, 8, 4>

<[, 8, 7>

<i, 8, 8>

<], 8, 9>

<++, 8, 10>

<key, 8, 12>

<), 8, 15>

<return, 9, 1>

<true, 9, 8>

<;, 9, 12>

<else, 10, 1>

<return, 10, 6>

<false, 10, 13>

<;, 10, 18>

<}, 11, 1>

<}, 12, 1>

<void, 13, 1>

<main, 13, 6>

<(, 13, 10>

<), 13, 11>

<{, 14, 1>

<int, 15, 1>

<a, 15, 5>

<[, 15, 6>

<20, 15, 7>

<], 15, 9>

<,, 15, 10>

<i, 15, 11>

<,, 15, 12>

<sum, 15, 13>

<;, 15, 16>

<bool, 16, 1>

<status, 16, 6>

<;, 16, 12>

<printf, 17, 1>

<(, 17, 7>

<"Enter array elements:", 17, 8>

<), 17, 29>

<;, 17, 30>

<for, 18, 1>

<(, 18, 4>

<i, 18, 5>

<=, 18, 6>

<0, 18, 7>

<;, 18, 8>

<i, 18, 9>

<<, 18, 10>

<10, 18, 11>

<;, 18, 13>

<++, 18, 14>

<i, 18, 16>

<), 18, 17>

<scanf, 19, 1>

<(, 19, 6>

<"%d", 19, 7>

<,, 19, 9>

<a, 19, 11>

<[, 19, 12>

<i, 19, 13>

<], 19, 14>

<), 19, 15>

<;, 19, 16>

<sum, 20, 1>

<=, 20, 4>

<a, 20, 5>

<[, 20, 6>

<0, 20, 7>

<], 20, 8>

<+, 20, 9>

<a, 20, 10>

<[, 20, 11>

<4, 20, 12>

<], 20, 13>

<;, 20, 14>

<status, 21, 1>

<=, 21, 7>

<search, 21, 8>

<(, 21, 14>

<a, 21, 15>

<,, 21, 16>

<sum, 21, 17>

<), 21, 20>

<;, 21, 21>

<printf, 22, 1>

<(, 22, 7>

<"%d", 22, 8>

<,, 22, 10>

<status, 22, 11>

<), 22, 17>

<;, 22, 18>

<}, 23, 1>