LAB EXERCISES:

Program:

```
//RDP FOR GRAMMAR
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "la.h"
void Program();
void declaration();
void data_type();
void identifier_list();
void statement_list();
void APrime();
void BPrime();
void assign_stat();
void statement();
void expn();
void simple_expn();
void eprime();
void term();
void seprime();
void dprime();
void tprime();
void factor();
void decision_stat();
void looping_stat();
void relop();
void addop();
void mulop();

struct token t;
FILE *f;
int epsilon=0;

void invalid(){
     printf("-----------------ERROR!----------------\n");
     exit(0);
}
void valid(){
     printf("----------------SUCCESS!---------------\n");
     exit(0);
}
void error(char *c){
```

```
     printf("missing %s at row %d col %d\n",c,t.row,t.col);
     invalid();
     exit(0);
}

void ungetToken(FILE *f,struct token s){
     int i = strlen(t.lexeme);
     fseek(f,-i,SEEK_CUR);
     col = col - i;
}
void Program(){
     t = getNextToken(f);
     //printf("program %s\n",t.lexeme);
     if(strcmp(t.lexeme,"main")==0){
          t = getNextToken(f);
          //printf("program %s\n",t.lexeme);
          if(strcmp(t.lexeme,"(")==0){
               t = getNextToken(f);
               //printf("program %s\n",t.lexeme);
               if(strcmp(t.lexeme,")")==0){
                    t = getNextToken(f);
                    //printf("program %s\n",t.lexeme);
                    if(strcmp(t.lexeme,"{")==0){
                         declaration();
                         statement_list();
                         t = getNextToken(f);
                         if (strcmp(t.lexeme,"}") == 0)
                              return;
                         else{
                              printf("missing } at row %d col
%d\n",t.row,t.col);
                         }
                    }
                    else{
                         printf("missing { at row %d col
%d\n",t.row,t.col);
                         exit(0);
                    }
               }
               else{
                    printf("missing ) at row %d col
%d\n",t.row,t.col);
                    exit(0);
               }
          }
          else{
               printf("missing ( at row %d col %d\n",t.row,t.col);
               exit(0);
          }
     }
     else{
          printf("missing main at row %d col %d\n",t.row,t.col);
```

```
          exit(0);
     }
}

void declaration(){
     data_type();
     if(epsilon == 1){
          ungetToken(f, t);
          epsilon = 0;
          return;
     }
     identifier_list();
     if(epsilon == 1){
          ungetToken(f, t);
          epsilon = 0;
          return;
     }
     t = getNextToken(f);
     //printf("declaration %s\n",t.lexeme);
     if(strcmp(t.lexeme, ";")==0){
          declaration();
          return;
     }
     else {
          printf("missing ; at row %d col %d\n",t.row,t.col);
          exit(0);
     }
}

void data_type(){
     t = getNextToken(f);
     //printf("data_type %s\n",t.lexeme);
     if(strcmp(t.lexeme, "int")==0 || strcmp(t.lexeme, "char")==0)
{
          epsilon = 0;
          return;
     }
     else{
          epsilon = 1;
          return;
     }
}

void identifier_list(){
     t = getNextToken(f);
     //printf("identifier_list %s\n",t.lexeme);
     if(strcmp(t.lexeme,"id")==0){
          APrime();
          if(epsilon == 1){
               ungetToken(f, t);
               epsilon = 0;
               return;
```

```
          }
          return;
     }
     else{
          printf("missing id at row %d col %d\n",t.row,t.col);
          exit(0);
     }
}

void APrime(){
     t = getNextToken(f);
     //printf("APrime %s\n",t.lexeme);
     if(strcmp(t.lexeme, ",")==0){
          identifier_list();
          return;
     }
     else if(strcmp(t.lexeme, "[")==0){
          t = getNextToken(f);
          //printf("APrime %s\n",t.lexeme);
          if(strcmp(t.lexeme, "num")==0){
               t = getNextToken(f);
               //printf("APrime %s\n",t.lexeme);
               if(strcmp(t.lexeme, "]")==0){
                    BPrime();
                    return;
               }
               else{
                    printf("missing ] at row %d col
%d\n",t.row,t.col);
                    exit(0);
               }
          }
          else error("number");
     }
     else {
          epsilon=1;
          //ungetToken(f, t);
          return;
     }
}

void BPrime(){
     t = getNextToken(f);
     //printf("BPrime %s\n",t.lexeme);
     if(strcmp(t.lexeme, ",")==0){
          identifier_list();
          return;
     }
     else {
          epsilon=1;
          return;
     }
```

```
}

void statement_list(){
     //printf("statement_list\n");
     statement();
     if(epsilon==1){
          ungetToken(f,t);
          epsilon=0;
          return;
     }
     statement_list();
}

void statement(){
     //printf("statement\n");
     assign_stat();
     //printf("statement from assign_stat\n");
     if(epsilon==1){
          ungetToken(f,t);
          epsilon=0;
          decision_stat();
          if(epsilon==1){
               ungetToken(f,t);
               epsilon=0;
               looping_stat();
               if(epsilon == 1){
                    return;
               }
               else
                    return;

          }
          else
               return;
     }
     t = getNextToken(f);
     //printf("statement %s\n",t.lexeme);
     if(strcmp(t.lexeme, ";")==0){
          epsilon=0;
          return;
     }
     else error(";");
}

void assign_stat(){
     t = getNextToken(f);
     //printf("assign_stat %s\n",t.lexeme);
     if(strcmp(t.lexeme,"id")==0){
          t = getNextToken(f);
          //printf("assign_stat %s\n",t.lexeme);
          if(strcmp(t.lexeme,"=")==0){
               expn();
```

```
          }
          else{
               error("=");
          }
     }
     else{
          epsilon = 1;
          return;
     }
}

void expn(){
     //printf("expn\n");
     simple_expn();
     eprime();
}

void eprime(){
     //printf("eprime\n");
     relop();
     if(epsilon==1){
          ungetToken(f,t);
          epsilon=0;
          return;
     }
     simple_expn();
}

void  simple_expn(){
     //printf("simple_expn\n");
     term();
     seprime();
}

void seprime(){
     //printf("seprime\n");
     //printf("seprime %s\n",t.lexeme);
     addop();
     if(epsilon==1){
          ungetToken(f,t);
          epsilon=0;
          return;
     }
     term();
     seprime();
}

void term(){
     //printf("term\n");
     factor();
     tprime();
}
```

```
void tprime(){
     //printf("tprime\n");
     //printf("tprime %s\n",t.lexeme);
     mulop();
     if(epsilon==1){
          ungetToken(f,t);
          epsilon=0;
          return;
     }
     factor();
     tprime();
}

void factor(){
     t = getNextToken(f);
     //printf("factor %s\n",t.lexeme);
     if(strcmp(t.lexeme, "id")==0){
          return;
     }
     else if(strcmp(t.lexeme, "num")==0){
          return;
     }
     else error("id/num");
}

void decision_stat(){
     t = getNextToken(f);
     //printf("decision_stat %s\n",t.lexeme);
     if(strcmp(t.lexeme, "if")==0){
          t = getNextToken(f);
          //printf("decision_stat %s\n",t.lexeme);
          if(strcmp(t.lexeme, "(")==0){
               expn();
               t = getNextToken(f);
               //printf("decision_stat %s\n",t.lexeme);
               if(strcmp(t.lexeme, ")")==0){
                    t = getNextToken(f);
                    //printf("decision_stat %s\n",t.lexeme);
                    if(strcmp(t.lexeme, "{")==0){
                         statement_list();
                         t = getNextToken(f);
                         //printf("decision_stat z %s\n",t.lexeme);
                         if(strcmp(t.lexeme, "}")==0){
                              dprime();
                              if(epsilon == 1){
                                   ungetToken(f, t);
                                   epsilon = 0;
                                   return;
                              }
                              return;
                         }
```

```
                    else error("}");
               }
               else error("{");
          }
          else error(")");
     }
     else error("(");
}
else {
     epsilon=1;
     return;
}
}

void dprime(){
     t = getNextToken(f);
     //printf("dprime %s\n",t.lexeme);
     if(strcmp(t.lexeme, "else")==0){
          t = getNextToken(f);
          //printf("dprime %s\n",t.lexeme);
          if(strcmp(t.lexeme, "{")==0){
               statement_list();
               t = getNextToken(f);
               //printf("dprime %s\n",t.lexeme);
               if(strcmp(t.lexeme, "}")==0){
                    return;
               }
               else error("}");
          }
          else error("{");
     }
     else {
          epsilon = 1;
          return;
     }
}

void looping_stat(){
     t = getNextToken(f);
     //printf("looping_stat %s\n",t.lexeme);
     if(strcmp(t.lexeme, "while")==0){
          t = getNextToken(f);
          //printf("looping_stat %s\n",t.lexeme);
          if(strcmp(t.lexeme, "(")==0){
               expn();
               t = getNextToken(f);
               if(strcmp(t.lexeme, ")")==0){
                    t = getNextToken(f);
                    //printf("looping_stat %s\n",t.lexeme);
                    if(strcmp(t.lexeme, "{")==0){
                         statement_list();
                         t = getNextToken(f);
```

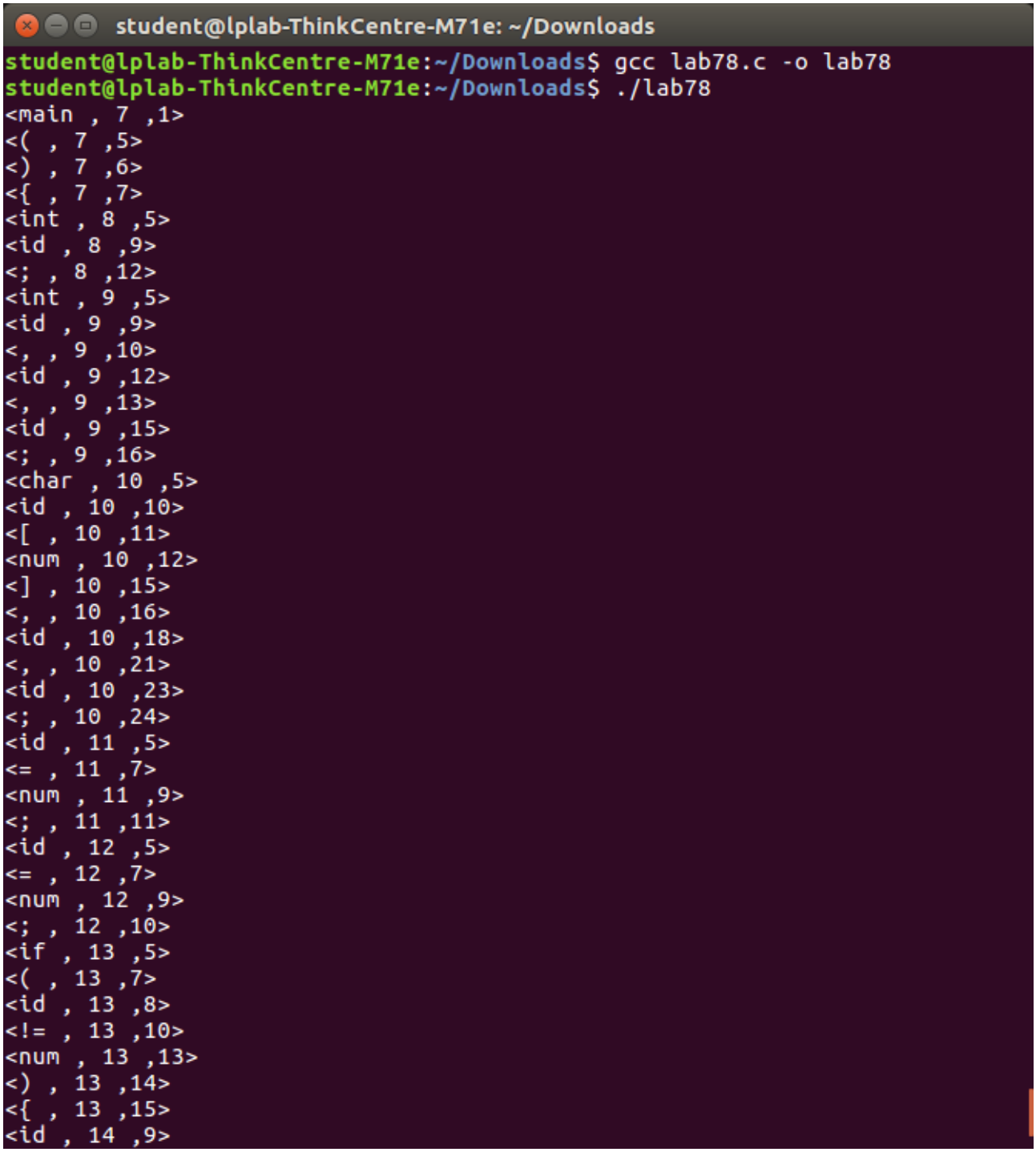
```
                    //printf("looping_stat %s\n",t.lexeme);
                    if(strcmp(t.lexeme, "}")==0){
                         return;
                    }
                    else error("}");
               }
               else error("{");
          }
          else error(")");
     }
     else error("(");
}
else if(strcmp(t.lexeme, "for")==0){
     t = getNextToken(f);
     if(strcmp(t.lexeme, "(")==0){
          assign_stat();
          if(epsilon==1){
               epsilon = 0;
               error("id");
          }
          t = getNextToken(f);
          if(strcmp(t.lexeme, ";")==0){
               expn();
               t = getNextToken(f);
               if(strcmp(t.lexeme, ";")==0){
                    assign_stat();
                    if(epsilon==1){
                         epsilon = 0;
                         error("id");
                    }
                    t = getNextToken(f);
                    if(strcmp(t.lexeme, ")")==0){
                         t = getNextToken(f);
                         if(strcmp(t.lexeme, "{")==0){
                              statement_list();
                              t = getNextToken(f);
                              if(strcmp(t.lexeme, "}")==0){
                                   return;
                              }
                              else error("}");
                         }
                         else error("{");
                    }
                    else error(")");
               }
               else error(";");
          }
          else error(";");
     }
     else error("(");
}
else{
```

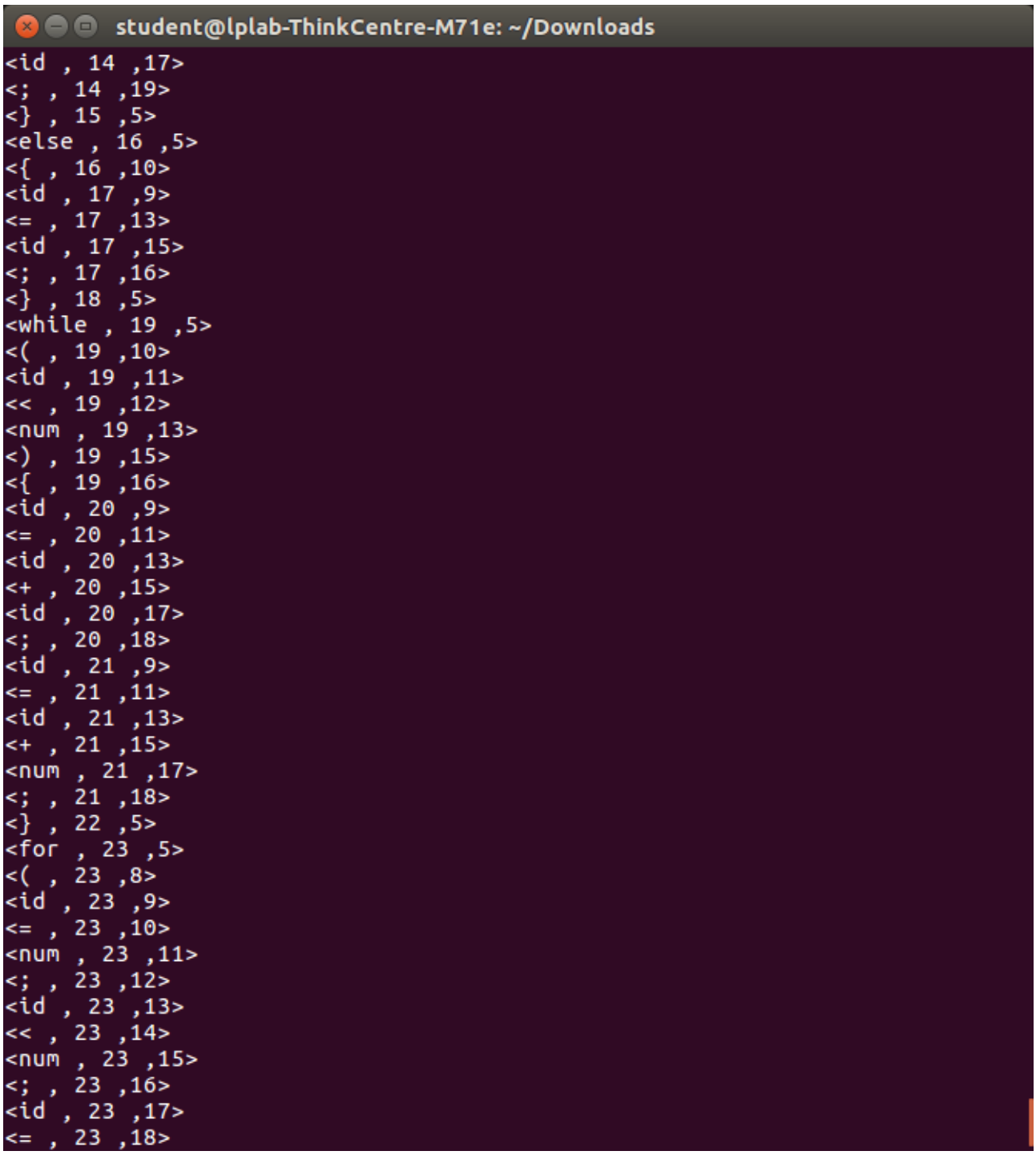
```
          epsilon = 1;
          return;
     }
}

void relop(){
     t = getNextToken(f);
     //printf("relop %s\n",t.lexeme);
     if(strcmp(t.lexeme, "==")==0){
          return;
     }
     else if(strcmp(t.lexeme, "!=")==0){
          return;
     }
     else if(strcmp(t.lexeme, "<=")==0){
          return;
     }
     else if(strcmp(t.lexeme, ">=")==0){
          return;
     }
     else if(strcmp(t.lexeme, ">")==0){
          return;
     }
     else if(strcmp(t.lexeme, "<")==0){
          return;
     }
     else {
          epsilon = 1;
          return;
     }
}

void addop(){
     t = getNextToken(f);
     //printf("addop %s\n",t.lexeme);
     if(strcmp(t.lexeme, "+")==0){
          return;
     }
     else if(strcmp(t.lexeme, "-")==0){
          return;
     }
     else {
          epsilon = 1;
          return;
     }
}

void mulop(){
     t = getNextToken(f);
     //printf("mulop %s\n",t.lexeme);
     if(strcmp(t.lexeme, "*")==0){
          return;
```

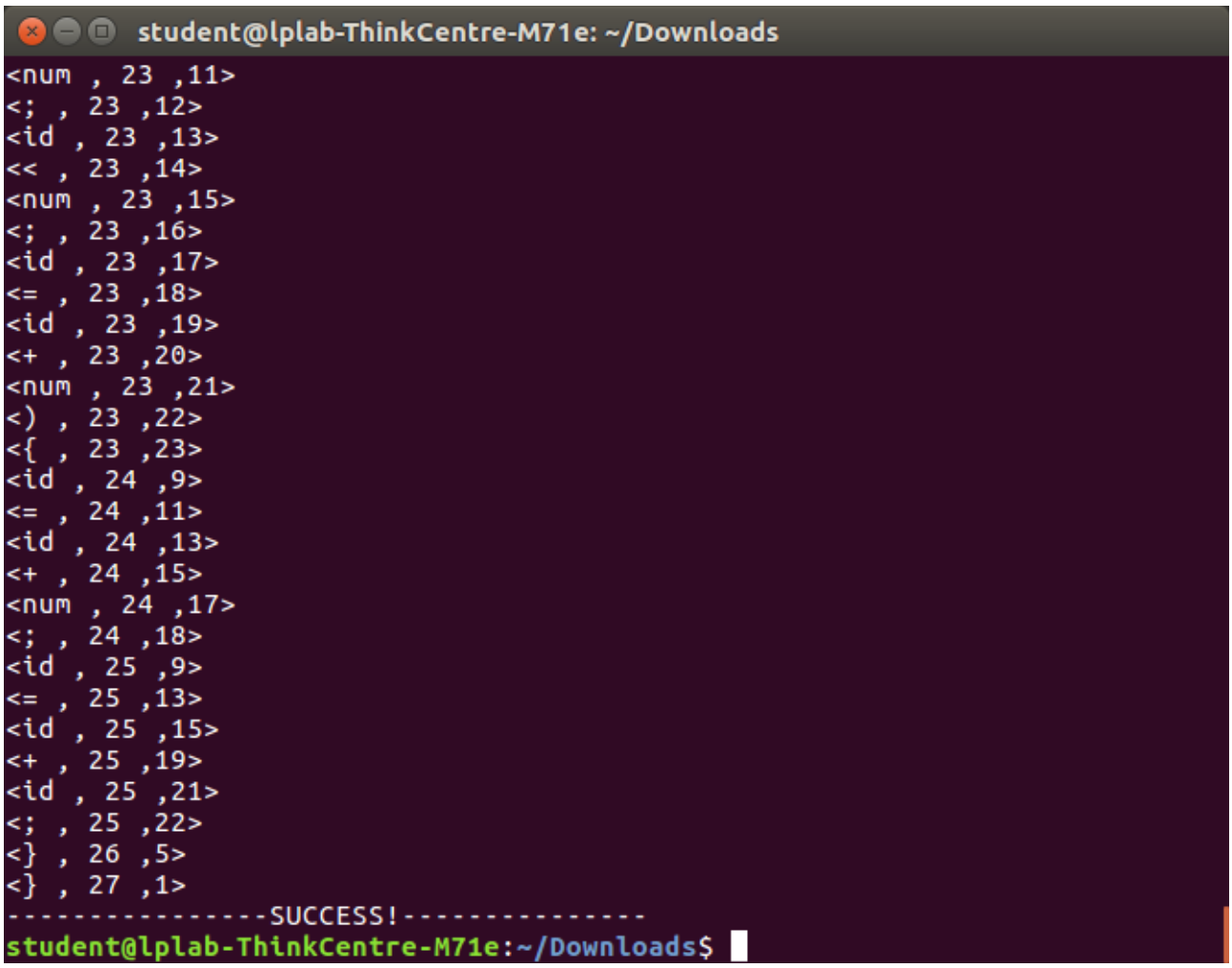
```
     }
     else if(strcmp(t.lexeme, "/")==0){
          return;
     }
     else if(strcmp(t.lexeme, "%")==0){
          return;
     }
     else {
          epsilon = 1;
          return;
     }
}

int main(){
     f = fopen("some.c","r");
     Program();
     fseek(f,0,SEEK_SET);
     row=1;
     while((t=getNextToken(f)).row!=-1)
          printf("<%s , %d ,%d>\n",t.lexeme,t.row,t.col);
          valid();

}
```

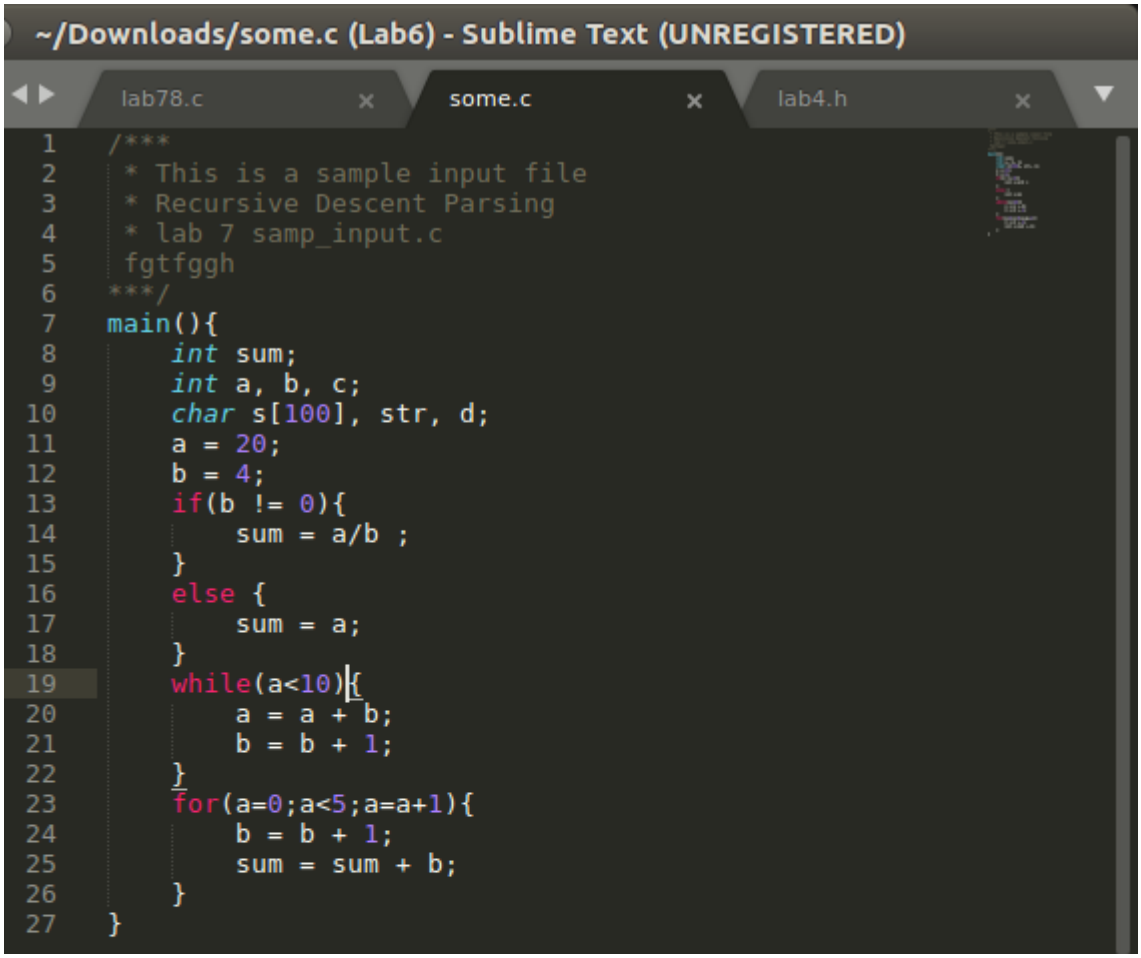
Output:

//Successful Case




student@lplab-ThinkCentre-M71e: ~/Downloads
<id , 14 ,17>
<; , 14 ,19>
<} , 15 ,5>
<else , 16 ,5>
<{ , 16 ,10>
<id , 17 ,9>
<= , 17 ,13>
<id , 17 ,15>
<; , 17 ,16>
<} , 18 ,5>
<while , 19 ,5>
<(, 19 ,10>
<id , 19 ,11>
<< , 19 ,12>
<num , 19 ,13>
<) , 19 ,15>
<{ , 19 ,16>
<id , 20 ,9>
<= , 20 ,11>
<id , 20 ,13>
<+ , 20 ,15>
<id , 20 ,17>
<; , 20 ,18>
<id , 21 ,9>
<= , 21 ,11>
<id , 21 ,13>
<+ , 21 ,15>
<num , 21 ,17>
<; , 21 ,18>
<} , 22 ,5>
<for , 23 ,5>
<(, 23 ,8>
<id , 23 ,9>
<= , 23 ,10>
<num , 23 ,11>
<; , 23 ,12>
<id , 23 ,13>
<< , 23 ,14>
<num , 23 ,15>
<; , 23 ,16>
<id , 23 ,17>
<= , 23 ,18>


student@lplab-ThinkCentre-M71e: ~/Downloads
<num , 23 ,11>
<; , 23 ,12>
<id , 23 ,13>
<< , 23 ,14>
<num , 23 ,15>
<; , 23 ,16>
<id , 23 ,17>
<= , 23 ,18>
<id , 23 ,19>
<+ , 23 ,20>
<num , 23 ,21>
<) , 23 ,22>
<{ , 23 ,23>
<id , 24 ,9>
<= , 24 ,11>
<id , 24 ,13>
<+ , 24 ,15>
<num , 24 ,17>
<; , 24 ,18>
<id , 25 ,9>
<= , 25 ,13>
<id , 25 ,15>
<+ , 25 ,19>
<id , 25 ,21>
<; , 25 ,22>
<} , 26 ,5>
<} , 27 ,1>
----------------SUCCESS!---------------
student@lplab-ThinkCentre-M71e:~/Downloads$


~/Downloads/some.c (Lab6) - Sublime Text (UNREGISTERED)
lab78.c
some.c
lab4.h
/***
* This is a sample input file
* Recursive Descent Parsing
* lab 7 samp_input.c
fgtfggh
***/
main(){
int sum;
int a, b, c;
char s[100], str, d;
a = 20;
b = 4;
if(b != 0){
sum = a/b ;
}
else {
sum = a;
}
while(a<10){
a = a + b;
b = b + 1;
}
for(a=0;a<5;a=a+1){
b = b + 1;
sum = sum + b;
}
}

//Unsuccessful Cases

```
student@lplab-ThinkCentre-M71e: ~/Downloads
student@lplab-ThinkCentre-M71e:~/Downloads$ gcc lab78.c -o lab78
student@lplab-ThinkCentre-M71e:~/Downloads$ ./lab78
missing ) at row 19 col 15
-----------------ERROR!----------------
```

~/Downloads/some.c (Lab6) - Sublime Text (UNREGISTERED)

lab78.c | some.c | lab4.h

```c
/***
 * This is a sample input file
 * Recursive Descent Parsing
 * lab 7 samp_input.c
 fgtfggh
***/
main(){
    int sum;
    int a, b, c;
    char s[100], str, d;
    a = 20;
    b = 4;
    if(b != 0){
        sum = a/b ;
    }
    else {
        sum = a;
    }
    while(a<10{
        a = a + b;
        b = b + 1;
    }
    for(a=0;a<5;a=a+1){
        b = b + 1;
        sum = sum + b;
    }
}
```

```
student@lplab-ThinkCentre-M71e:~/Downloads$ gcc lab78.c -o lab78
student@lplab-ThinkCentre-M71e:~/Downloads$ ./lab78
missing ; at row 15 col 5
-----------------ERROR!----------------
student@lplab-ThinkCentre-M71e:~/Downloads$
```

~/Downloads/some.c (Lab6) - Sublime Text (UNREGIS

lab78.c | some.c

```c
/***
 * This is a sample input file
 * Recursive Descent Parsing
 * lab 7 samp_input.c
 fgtfggh
***/
main(){
    int sum;
    int a, b, c;
    char s[100], str, d;
    a = 20;
    b = 4;
    if(b != 0){
        sum = a/b
    }
    else {
        sum = a;
    }
    while(a<10){
        a = a + b;
        b = b + 1;
    }
    for(a=0;a<5;a=a+1){
        b = b + 1;
        sum = sum + b;
    }
}
```

```
//La.h
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define ipname "file.c"
#define op1name "output1.c"
#define op2name "output2.c"
struct token{
     char lexeme[64];
     unsigned int row,col;
};

struct symtable
{
    int sno;
    char lex[128];
    char dtype[64];
    int size;
};

static int row=1,col=1,inFunc=0;
char buf[2048];
char dbuf[128];
char w[25];
int ind=0,i=0,j=0,flag=0;
int tabsz[10];
struct symtable st[10];


char spsym[] = {'?',':',';','.',','};
const char *keywords[] = {"const", "bool", "char", "int", "float",
"double", "unsigned", "return", "for", "while", "do", "switch",
"if", "else", "case", "break",
"continue","return","scanf","short","const","sizeof","true","false
","typedef","main"};
char ariop[]={'*','%','/','+','-'};
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 iskword(char *w){
     for(int I=0;I<sizeof(keywords)/sizeof(char*);I++){
          if(strcmp(w,keywords[I])==0){
               return 1;
          }
     }
     return 0;
}



int findTable(char *nam,int n){
    int I=0;
    for(I=0;I<n;I++){
        if(strcmp(st[I].lex,nam)==0){
            return 1;
        }
    }
    return 0;
}

struct symtable fillTable(int x,char *lexn,char *dt,int s)
{
    st[ind].sno=x;
    strcpy(st[ind].lex,lexn);
    strcpy(st[ind].dtype,dt);
    st[ind].size=s;
}

void printTable(int n)
{
    for(int I=0;I<n;I++)
    {
        printf("%d\t%s\t\t%s\t
%d\n",st[I].sno,st[I].lex,st[I].dtype,st[I].size);

printf("_____________________________________________________\n\n");
    }
}

int isop_sym(char c , char *class){
     int n=0;
     if(class==spsym){
          n = sizeof(spsym)/sizeof(char);
     }
     else if(class==ariop){
          n = sizeof(ariop)/sizeof(char);
     }
     for(int I=0;I<n;I++){
          if(c==class[I]){
```

```
                return 1;
            }
        }
        return 0;
}

void makeToken(struct token *t,char c, int row, int col){
        t->lexeme[0]=c;
        t->lexeme[1]='\0';
        t->row=row;
        t->col=col;
}

void newLine() {
        row++;
        col=1;
}

int dt(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* fp){
        int c;
        struct token Token= {
            .row=-1
        };
        int gotToken=0;
        while(!gotToken &&(c=fgetc(fp))!=EOF){
       if(c=='/')
           {
               char d = fgetc(fp);
               if (d == '/')
               {
                   while(c != '\n')
                       c = getc(fp);
                   newLine();
               }
               else if (d == '*')
               {
                   do
```

```
        {
            while(c != '*')
            {
                if(c=='\n')
                    newLine();
                c = getc(fp);
            }
            c = getc(fp);
        } while (c != '/');
    }
    else
    {
        fseek(fp,-1,SEEK_CUR);
        makeToken(&Token,c,row,col);
        gotToken=1;
        ++col;
    }
}

// Ignoring predirectives
else if(c=='#' && inFunc==0)
{
    char inc[50]="#include",def[50]="#define",d='#';
    int i=0,isDirective=0;
    while(d==inc[i] && inc[i]!='\0')
    {
        d=fgetc(fp);
        i++;
    }
    if(inc[i]=='\0')
    {
        isDirective=1;
    }
    fseek(fp,-i,SEEK_CUR);
    d='#';
    i=0;
    while(d==def[i] && def[i]!='\0')
    {
        d=fgetc(fp);
        i++;
    }
    if(def[i]=='\0')
    {
        isDirective=1;
    }
    fseek(fp,-i,SEEK_CUR);
    if(isDirective)
    {
        while(c!='\n')
        {
            c=fgetc(fp);
        }
```

```
            newLine();
          }
      }
        else if(isop_sym(c,spsym)){
             makeToken(&Token,c,row,col);
             gotToken=1;
             ++col;
        }
        else if(c=='('){
             makeToken(&Token,c,row,col);
             gotToken=1;
             ++col;
        }
        else if(c==')') {
             makeToken(&Token,c,row,col);
             gotToken=1;
             ++col;
        }
        else if(c=='{') {
             makeToken(&Token,c,row,col);
          gotToken=1;
          ++col;
          flag++;
      }
  else if(c=='}') {
        makeToken(&Token,c,row,col);
        gotToken=1;
        ++col;
        flag--;
        if(flag==0){
        tabsz[i]=j;
        i++;
        j=0;

        }
  }
  else if(c=='['){
        makeToken(&Token,c,row,col);
        gotToken=1;
        col++;
  }
  else if(c==']'){
        makeToken(&Token,c,row,col);
        gotToken=1;
        col++;

  }
  else if(c=='+'){
        int d = fgetc(fp);
        if(d=='+'){
        Token.lexeme[0]='+';
        Token.lexeme[1]='+';
```

```
      Token.lexeme[2]='\0';
      Token.row=row;
      Token.col=col++;
 ++col;

 gotToken=1;
 }
  else{
    makeToken(&Token,c,row,col);
      gotToken=1;
      ++col;
      fseek(fp,-1,SEEK_CUR);
  }
}
else if(c=='-'){
      char d = fgetc(fp);
 if(d=='-'){
      Token.lexeme[0]=Token.lexeme[1]='-';
      Token.lexeme[2]='\0';
      Token.row=row;
      Token.col=col++;
      gotToken=1;
  }
      else{
      makeToken(&Token,c,row,col);
           gotToken=1;
      ++col;
           fseek(fp,-1,SEEK_CUR);
      }
 }
else if(c=='='){
 char d = fgetc(fp);
      if(d=='='){
           Token.lexeme[0]=Token.lexeme[1]='=';
      Token.lexeme[2]='\0';
      Token.row=row;
      Token.col=col++;
      gotToken=1;
      }
 else{
           makeToken(&Token,c,row,col);
           gotToken=1;
           ++col;
           fseek(fp,-1,SEEK_CUR);
      }
 }
else if(isop_sym(c,ariop)){
      makeToken(&Token,c,row,col);
      gotToken=1;
      ++col;
 }
else if(isdigit(c)){
```

```
 Token.row = row;
 Token.col = col++;
 //Token.lexeme[0]=c;
 int k=1;
      while((c=fgetc(fp))!=EOF && isdigit(c)) {
           //Token.lexeme[k++]=c;
      col++;
      }
 strcpy(Token.lexeme,"num");
 gotToken=1;
 fseek(fp,-1,SEEK_CUR);
 }
else if(c=='\n') {
      newLine();
}
else if(c=='\t') {
      col=col+1;
 }
else if(c==' ') {
 ++col;
}
else if(isalpha(c)||c=='_') {
 char m[10];
 int s;
      Token.row=row;
      Token.col=col++;
      Token.lexeme[0]=c;
      int k=1;
      while((c=fgetc(fp))!= EOF && isalnum(c)) {
           Token.lexeme[k++]=c;
           ++col;
      }
      Token.lexeme[k]='\0';
      fseek(fp,-1,SEEK_CUR);

      if(iskword(Token.lexeme)) {
      if(isdtype(Token.lexeme)==1){
            strcpy(dbuf,Token.lexeme);
            strcpy(w,Token.lexeme);
        }
      ;
 }
      else{
      char d = fgetc(fp);
      int n=1;
      int k=0;
      if(d=='['){
                while((c=fgetc(fp))!=EOF && c!=']'){
                n++;
           if(isdigit(c)){m[k++]=c;}
           }
           m[k]='\0';
```

```
            n++;
            if(findTable(Token.lexeme,ind)==0){
            s=atoi(m);
            fillTable(ind,Token.lexeme,dbuf,dt(dbuf)*s);
            ind++;
            }
                 fseek(fp,-n,SEEK_CUR);
      }
      else if(d=='('){
            if(findTable(Token.lexeme,ind)==0){
                 fillTable(ind,Token.lexeme,"func",-1);
                 ind++;
                 }
                 fseek(fp,-n,SEEK_CUR);
      }
      else{
            if(findTable(Token.lexeme,ind)==0){
                 fillTable(ind,Token.lexeme,dbuf,dt(dbuf));
                 ind++;
        }
            fseek(fp,-n,SEEK_CUR);
      }
      strcpy(Token.lexeme,"id");
      }
            gotToken=1;

 }
 else if(c == '"') {
      Token.row = row;
      Token.col = col++;
      strcpy(Token.lexeme, "STRINGLITERAL");
      int k = 1;
      while((c = fgetc(fp)) != EOF && c != '"') {
      ++col;
    }
            ++col;
      gotToken = 1;
}
else if(c == '<' || c == '>' || c == '!') {
      makeToken(&Token, c,row, col);
      ++col;
      int d = fgetc(fp);
      if(d == '=') {
      ++col;
      strcat(Token.lexeme, "=");
      }
      else{
      if(c == '!'){
      }
      fseek(fp, -1, SEEK_CUR);
      }
      gotToken = 1;
```

```
     }
    else if(c == '&' || c == '|') {
          int d = fgetc(fp);
          if(c == d) {
          Token.lexeme[0] = Token.lexeme[1] = c;
          Token.lexeme[2] = '\0';
          Token.row = row;
          Token.col = col;
          ++col;
          gotToken = 1;
     }
     else {
          Token.lexeme[0] = c;
          Token.lexeme[1] = '\0';
          Token.row = row;
          Token.col = col;
          ++col;
          gotToken = 1;
          fseek(fp, -1, SEEK_CUR);
          }
     ++col;
     }        // going to next column if unexpected literal gotten
    else if(c == '$') {
          Token.lexeme[0] = '$';
          Token.lexeme[1] = '\0';
          Token.row = row;
          Token.col = col;
          ++col;
          gotToken = 1;
     }
    else {
      ++col;
    }
}
     //printf("TOKEN %s\n",Token.lexeme);
    return Token;
}
```