CD Lab 7 and 8

Aniruddha Amit Dutta

180905488

Roll -58

Q1.

// GET next token OR la.c file

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
#include <errno.h>


#define SZ 20

struct token{
  char tok_type[SZ];
  char lexeme[SZ];
  int row,col,idx;
  int sz;
};

struct ListElement{
  struct token tok;
  struct ListElement *next;
};

struct ListElement *TABLE[SZ];
int row=1,col=1,val=-1,TableLength = 0;
char dbuff[SZ];
bool FILENOTENDED=true;

char keyword[34][10]={"printf","scanf","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"};


bool iskeyword(char* buf){
   for(int i=0;i<34;i++){
          if(strcmp(keyword[i],buf)==0)
                 return true;
   }
   return false;
}

bool isDelimiter(char ch){
          if (ch == ',' || ch == ';'  || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch ==
'{' || ch == '}')
        return true;
    return false;
}

bool isArithmetic_operator(char ch)
{
    if (ch == '%' || ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' )
        return true;
    return false;
}

void printtok(struct token t){
   printf("<%s,%d,%d> ",t.lexeme,t.row,t.col-1);
}

int SEARCH(struct token tk){
    //printf("s\n");
    struct ListElement * cur;
    for(int i=0;i<=val;i++){
      cur = TABLE[i];
     if(cur&&strcmp(tk.tok_type,"func")==0){
           if(strcmp((cur->tok).lexeme,tk.lexeme)==0){
            return 1;
          }
      }
     else{
       while(cur){
         if(strcmp((cur->tok).lexeme,tk.lexeme)==0&&strcmp((cur-
>tok).tok_type,tk.tok_type)==0&&(cur->tok).idx==tk.idx){
         return 1;
       }
       cur=cur->next;
      }
    }
   }
    return 0;
}

void INSERT(struct token tk){
  if(strcmp(tk.tok_type,"func")!=0&&SEARCH(tk)==1){
     return;
   }

  struct ListElement* cur = malloc(sizeof(struct ListElement));
  cur->tok = tk;
  cur->next = NULL;

  if(TABLE[val]==NULL){
     TABLE[val] = cur; // No collosion.
  }
  else{
    struct ListElement * ele= TABLE[val];
    while(ele->next!=NULL){
       ele = ele->next; // Add the element at the End in the case of a collision.
    }
    ele->next = cur;
  }
}

void sanitize(struct token s,FILE * fa){
    int len;
        if(s.lexeme[0]=='i'&&s.lexeme[1]=='d'&&s.lexeme[2]==' ')
           len=strlen(s.lexeme)-3;
        else len=strlen(s.lexeme);
        fseek(fa,-1*len,SEEK_CUR);
}
```

```
struct token getnextToken(FILE *fa){
    char ca,cb;
    int i,j;
    char buf[SZ],temp[SZ];
    struct token s;
    ca=fgetc(fa);
    while(ca!=EOF){
         //newline
         if(ca=='\n'){
                  row++;
                  col=1;
                  //printf("\n");
         }
         //blank space and tabs
         else if(ca==' '||ca=='\t'){
                    col++;
              while(ca==' '||ca=='\t')
                   ca=fgetc(fa);
              fseek(fa,-1,SEEK_CUR);
         }
        //comments
        else if(ca=='/'){
            col++;
         cb=fgetc(fa);
         if(cb=='/'){
                while(ca!='\n')
                 ca=fgetc(fa);
                fseek(fa,-1,SEEK_CUR);
         }
         else if(cb=='*'){
                   do{
                          while(ca!='*')
                                   ca = fgetc(fa);
                          ca = fgetc(fa);
                   }while(ca!='/');
         }
         else{
                  i=0;
                while(ca!='\n'){
                   temp[i++] = ca;
                   ca = fgetc(fa);
                }
            temp[i]='\0';
            strcpy(s.lexeme,"syntax error");
            s.row=row;
            s.col=col;
            fseek(fa,-1,SEEK_CUR);
            return s;
         }
        }
        //preprocessor
        else if(ca=='#'){
         i=0;
           while(ca!='\n'){
          temp[i++]=ca;
          ca=fgetc(fa);
           }
           temp[i]='\0';
           fseek(fa,-1,SEEK_CUR);
           if(strstr(temp,"#include")==NULL && strstr(temp,"#define")==NULL)
{//not working
            printf("include\n");
              strcpy(s.lexeme,"syntax error");
            s.row=row;
            // row++;
            s.col=col;
            return s;
           }
        }
        //keywords and identifiers
        else if(isalpha(ca)||ca=='_'){
         i=0;
         while(isalnum(ca)||ca=='_'){
              buf[i++]=ca;
              ca=fgetc(fa);
              col++;
         }
         buf[i]='\0';
         fseek(fa,-1,SEEK_CUR);

         if(iskeyword(buf)){
            strcpy(s.lexeme,buf);
            strcpy(dbuff,buf);
            s.row=row;
```

```
                s.col=col-strlen(buf)+1;
                return s;
            }
            else{
                 if(ca=='('){
                        strcpy(s.lexeme,buf);
                        strcpy(s.tok_type,"func");
                        s.sz=-1;
                        if(SEARCH(s)==0){
                            val++;
                        }
                        s.idx = val;
                        INSERT(s);
                        return s;
                      }
                  char w[10]="";
                  strcat(w,"id ");
                  strcat(w,buf);
                  strcpy(s.lexeme,w);
                  strcpy(s.tok_type,dbuff);
                  s.row=row;
                  s.col=col-strlen(buf)+1;

                  if(strcmp(dbuff,"int")==0)
                     s.sz=sizeof(int);
                  else if(strcmp(dbuff,"char")==0)
                     s.sz=sizeof(char);
                  else if(strcmp(dbuff,"bool")==0)
                     s.sz=sizeof(bool);
                  else
                     s.sz=0;
                  if(strcmp(dbuff,"return")==0||strcmp(dbuff,"if")==0||
strcmp(dbuff,"scanf")==0||strcmp(dbuff,"printf")==0||strcmp(dbuff,"for")==0)
                     return s;
                  s.idx=val;
                  INSERT(s);

                  return s;
            }
           }
           //relational operator
           else if(ca=='='||ca=='>'||ca=='<'||ca=='!'){
```

```
 cb=fgetc(fa);
 i=0;
 temp[i++]=ca;
   col++;
 if(cb=='='){
        temp[i++] = cb;
        temp[i] = '\0';
        strcpy(s.lexeme,temp);
        s.row=row;
        s.col=col;
        col++;
        return s;
 }
 else{
           temp[i]='\0';
           strcpy(s.lexeme,temp);
          s.row=row;
          s.col=col;
           fseek(fa,-1,SEEK_CUR);
          return s;
 }

}
//string
else if(ca=='"'){
  i=0;
    do{
          col++;
          i++;
          ca=fgetc(fa);
    }while(ca!='"');
    col++;
    strcpy(s.lexeme,"string literal");
          s.row=row;
          s.col=col-i;
          return s;
}
//delimiters
else if(isDelimiter(ca)){
  i=0;
  temp[i++]=ca;
  temp[i]='\0';
```

```
          col++;
            strcpy(s.lexeme,temp);
                 s.row=row;
                 s.col=col;
                 return s;
        }
        //numeric constants
        else if(isdigit(ca)){
         i=0;
         while(isdigit(ca)){
                 col++;
                 i++;
                 ca=fgetc(fa);
         }
         fseek(fa,-1,SEEK_CUR);
         strcpy(s.lexeme,"num");
                 s.row=row;
                 s.col=col-i+1;
                 return s;
        }
        //arithmetic op
        else if(isArithmetic_operator(ca)){
          i=0;
          temp[i++]=ca;
          temp[i]='\0';
          col++;
            strcpy(s.lexeme,temp);
                 s.row=row;
                 s.col=col;
                 return s;
        }
        ca=fgetc(fa);
    }
    strcpy(s.lexeme,"end");
    return s;
}

void Initialize(){
  for(int i=0;i<SZ;i++){
      TABLE[i] = NULL;
  }
}
```

```
void Display(){
//iterate through the linked list and display
 for(int i=0;i<=val;i++){
    struct ListElement * cur = TABLE[i];
    printf("%d %s %s\n\n",i+1,(cur->tok).lexeme,(cur->tok).tok_type);
    cur=cur->next;
    while(cur){
        printf("%s   %s   %d\n",(cur->tok).lexeme, (cur->tok).tok_type,(cur-
>tok).sz);
        cur=cur->next;
    }
    printf("******************\n");
 }
}
```

// RDP parser

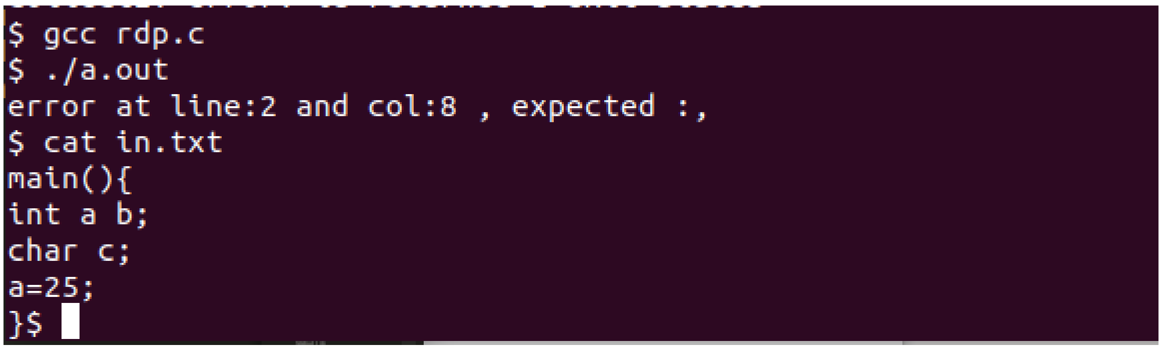
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
#include "la.c"

struct token s;
FILE * fa;

void declartions();
void assign_stat();
void assign_stat_prime();
bool datatype();
void identifier_list();
void identifier_list_prime();

void report_error(){
         printf("error at line:%d and col:%d , expected :",s.row,s.col );
}



void Program(){
	s=getnextToken(fa);
	// printf("token - %s\n",s.lexeme);
	if(!strcmp(s.lexeme,"main")){
		// printf("here %s\n","i am" );
		s=getnextToken(fa);
		if(!strcmp(s.lexeme,"(")){
			s=getnextToken(fa);
			if(!strcmp(s.lexeme,")") ){
				s=getnextToken(fa);
				if(!strcmp(s.lexeme,"{")){
					declartions();
					assign_stat();
					s=getnextToken(fa);
					if(!strcmp(s.lexeme,"}")){
						// printf("line 39\n");
						return ;
					}else{
						report_error(); printf("}\n");
						exit(0);
					}
				}else{
					report_error(); printf("{\n");
					exit(0);
				}
			}
			else{
				report_error(); printf(")\n");
				exit(0);
			}
		}else{
			report_error(); printf("(\n");
			exit(0);
		}

	}else{
		report_error(); printf("main\n");
		exit(0);
	}
}

void declartions(){
	s=getnextToken(fa);
	if(!strcmp(s.lexeme,"int") || !strcmp(s.lexeme,"char")){
		identifier_list();
		s=getnextToken(fa);
		if(!strcmp(s.lexeme,";")){
			declartions();

		}else{
			report_error(); printf(";\n");
			exit(0);
		}
	}else{
		sanitize(s,fa);
	}
}

void identifier_list(){
	s=getnextToken(fa);
	if((s.lexeme[0]=='i'&&s.lexeme[1]=='d'&&s.lexeme[2]==' ')){
		identifier_list_prime();
	}else{
		report_error();printf("identifier \n");
		exit(0);
	}
}

void identifier_list_prime(){
	s=getnextToken(fa);
	if(!strcmp(s.lexeme,",")){
		identifier_list();
	}else if(!strcmp(s.lexeme,";")){
		sanitize(s,fa);
	}else{
		report_error(); printf(",\n");
		exit(0);
	}
}

void assign_stat(){
	s=getnextToken(fa);
	if((s.lexeme[0]=='i'&&s.lexeme[1]=='d'&&s.lexeme[2]==' ')){
		s=getnextToken(fa);
		if(!strcmp(s.lexeme,"=")){
```

```
                assign_stat_prime();
            }
            else{
                report_error(); printf("=\n");
                exit(0);
            }
        }else{
            report_error(); printf("identifier \n");
            exit(0);
        }
}

void assign_stat_prime(){
        s=getnextToken(fa);
        if((s.lexeme[0]=='i'&&s.lexeme[1]=='d'&&s.lexeme[2]==' ') || !
strcmp(s.lexeme,"num") ){
            s=getnextToken(fa);
            if(!strcmp(s.lexeme,";")){
                return ;
            }else{
                report_error(); printf(";\n");
                exit(0);
            }
        }else{
            report_error(); printf(" identifier or number \n");
            exit(0);
        }
}

bool datatype(){
        s=getnextToken(fa);
        if(!strcmp(s.lexeme,"int") || !strcmp(s.lexeme,"char")){
            return 1;
        }else{
            report_error(); printf(" datatype \n");
            exit(0);
        }
}

int main(){

        fa = fopen("in.txt","r");
```

```
	if(fa==NULL){
		perror("fopen");
		exit(0);
	}
	Program();
	s=getnextToken(fa);
	if(!strcmp(s.lexeme,"end")){
		printf("\n successfully parsed \n");
	}
}
```

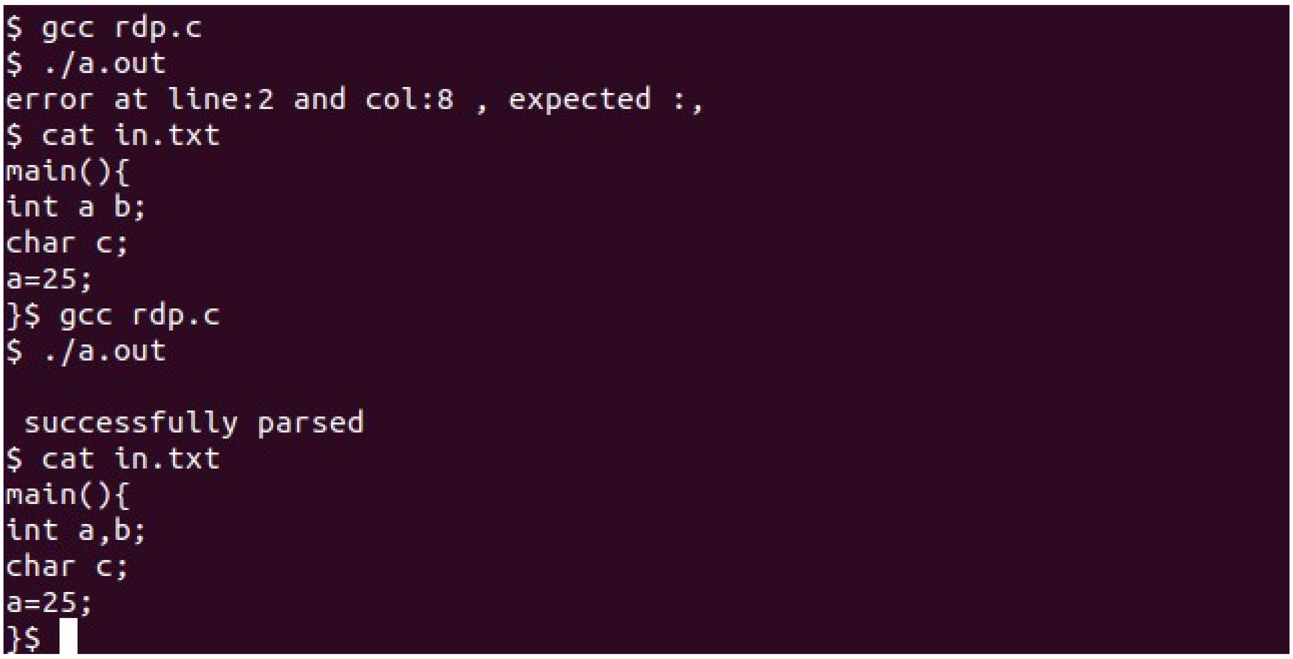
// **output**

error -



```
$ gcc rdp.c
$ ./a.out
error at line:2 and col:8 , expected :,
$ cat in.txt
main(){
int a b;
char c;
a=25;
}$
```

successful-



```
$ gcc rdp.c
$ ./a.out
error at line:2 and col:8 , expected :,
$ cat in.txt
main(){
int a b;
char c;
a=25;
}$ gcc rdp.c
$ ./a.out

 successfully parsed
$ cat in.txt
main(){
int a,b;
char c;
a=25;
}$
```

Q2.

// RDP parser

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
#include "la.c"

struct token s;
FILE * fa;

void declartions();
void assign_stat();
bool datatype();
void identifier_list();
void identifier_list_prime();
void identifier_list_prime_prime();
void statement();
void statement_list();
void expn();
void eprime();
void term();
void tprime();
void factor();
void relop();
void relop_prime();
void addop();
void mulop();
void simple_expn();
void seprime();

void report_error(){
        printf("\n\nerror at line:%d and col:%d , expected :",s.row,s.col );
}
```

```
void Program(){
        s=getnextToken(fa); printtok(s);
        // printf("token - %s\n",s.lexeme);
        if(!strcmp(s.lexeme,"main")){
                // printf("here %s\n","i am" );
                s=getnextToken(fa); printtok(s);
                if(!strcmp(s.lexeme,"(")){
                        s=getnextToken(fa); printtok(s);
                        if(!strcmp(s.lexeme,")") ){
                                s=getnextToken(fa); printtok(s);
                                if(!strcmp(s.lexeme,"{")){
                                        declartions();
                                        statement_list();
                                        s=getnextToken(fa); printtok(s);
                                        if(!strcmp(s.lexeme,"}")){
                                                // printf("line 39\n");
                                                return ;
                                        }else{
                                                report_error(); printf("}\n");
                                                exit(0);
                                        }
                                }else{
                                        report_error(); printf("{\n");
                                        exit(0);
                                }
                        }
                        else{
                                report_error(); printf(")\n");
                                exit(0);
                        }
                }else{
                        report_error(); printf("(\n");
                        exit(0);
                }

        }else{
                report_error(); printf("main\n");
                exit(0);
        }
}
```

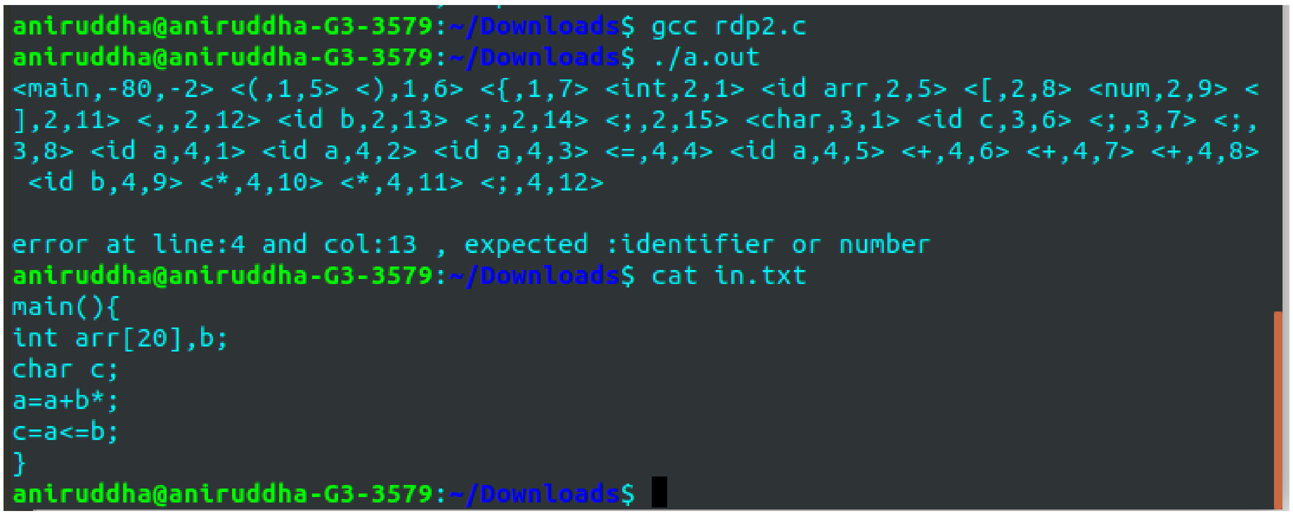
```
void declartions(){
        s=getnextToken(fa); printtok(s);
        if(!strcmp(s.lexeme,"int") || !strcmp(s.lexeme,"char")){
                identifier_list();
                s=getnextToken(fa); printtok(s);
                if(!strcmp(s.lexeme,";")){
                        declartions();

                }else{
                        report_error(); printf(";\n");
                        exit(0);
                }
        }else{
                sanitize(s,fa);
        }
}

void identifier_list(){
        s=getnextToken(fa); printtok(s);
        if(s.lexeme[0]=='i'&&s.lexeme[1]=='d'&&s.lexeme[2]==' '){
                identifier_list_prime();
        }
        else{
                report_error(); printf(" identifier \n");
                exit(0);
        }
}

void identifier_list_prime(){

        s=getnextToken(fa); printtok(s);
        if(!strcmp(s.lexeme,",")){
                identifier_list();
        }
        else if(!strcmp(s.lexeme,"[")){
                identifier_list_prime_prime();
        }
        else{
                sanitize(s,fa);
        }
}

void identifier_list_prime_prime(){
	s=getnextToken(fa); printtok(s);
	if(!strcmp(s.lexeme,"num")){
		s=getnextToken(fa); printtok(s);
		if(!strcmp(s.lexeme,"]")){
			s=getnextToken(fa); printtok(s);
			if(!strcmp(s.lexeme,","))
				identifier_list();
			else{
				sanitize(s,fa);
			}
		}
		else{
			report_error(); printf("]\n");
			exit(0);
		}
	}
}



void statement_list(){
	s=getnextToken(fa); printtok(s);
	if(s.lexeme[0]=='i'&&s.lexeme[1]=='d'&&s.lexeme[2]==' '){
		sanitize(s,fa);
		statement();
		statement_list();
	}
	else
		sanitize(s,fa);
}

void statement()
{
	assign_stat();
	s=getnextToken(fa); printtok(s);
	if(!strcmp(s.lexeme,";")){
		return ;
	}else{
		report_error(); printf(";\n");
		exit(0);
	}
}

void assign_stat(){
	s=getnextToken(fa); printtok(s);
	if(s.lexeme[0]=='i'&&s.lexeme[1]=='d'&&s.lexeme[2]==' '){
		s=getnextToken(fa); printtok(s);
		if(!strcmp(s.lexeme,"=")){
			expn();
		}else{
			report_error(); printf("=\n");
			exit(0);
		}
	}else{
		report_error(); printf("identifier \n");
		exit(0);
	}
}

void expn(){
	simple_expn();
	eprime();
}

void eprime(){
	s=getnextToken(fa); printtok(s);
	if(!strcmp(s.lexeme,"==")||!strcmp(s.lexeme,"!=")||!
strcmp(s.lexeme,"<=")||!strcmp(s.lexeme,">=")||!strcmp(s.lexeme,"<")||!
strcmp(s.lexeme,">")){
		sanitize(s,fa);
		relop();
		simple_expn();
	}
	else
		sanitize(s,fa);
}

void simple_expn(){
	term();
	seprime();
}
```

```
void seprime(){
        s=getnextToken(fa); printtok(s);
        if(!strcmp(s.lexeme,"+")||!strcmp(s.lexeme,"-")){
                sanitize(s,fa);
                addop();
                term();
                seprime();
        }
        else
                sanitize(s,fa);

}

void term(){
        factor();
        tprime();
}

void tprime(){
        s=getnextToken(fa); printtok(s);
        if(!strcmp(s.lexeme,"*")||!strcmp(s.lexeme,"/")||!strcmp(s.lexeme,"%")){
                sanitize(s,fa);
                mulop();
                factor();
                tprime();
        }
        else
                sanitize(s,fa);
}

void factor(){
        s=getnextToken(fa); printtok(s);
        if((s.lexeme[0]=='i'&&s.lexeme[1]=='d'&&s.lexeme[2]==' ')|| !
strcmp(s.lexeme,"num") ){
                return;
        }
        else{
                report_error(); printf("identifier or number \n");
                exit(0);
        }
}
```

```
void relop(){
        s=getnextToken(fa); printtok(s);
        if(!strcmp(s.lexeme,"==") || !strcmp(s.lexeme,"!=") || !
strcmp(s.lexeme,"<=") || !strcmp(s.lexeme,">=") || !strcmp(s.lexeme,"<") || !
strcmp(s.lexeme,">") ){
                return;
        }
        else{
                report_error(); printf(" relational operator \n");
                exit(0);
        }
}

void addop(){
        s=getnextToken(fa); printtok(s);
        if(strcmp(s.lexeme,"+")||strcmp(s.lexeme,"-")){
                return;
        }
        else{
                report_error(); printf(" + or - \n");
                exit(0);
        }
}

void mulop(){
        s=getnextToken(fa); printtok(s);
        if( !strcmp(s.lexeme,"*") || !strcmp(s.lexeme,"/") || !
strcmp(s.lexeme,"%")){
                return;
        }
        else{
                report_error(); printf(" multiply and divide operator\n");
                exit(0);
        }
}

int main(){

        fa = fopen("in.txt","r");
        if(fa==NULL){
                perror("fopen");
                exit(0);
        }
        Program();
        s=getnextToken(fa); printtok(s);
        if(!strcmp(s.lexeme,"end")){
                printf("\n successfully parsed \n");
        }
}
```

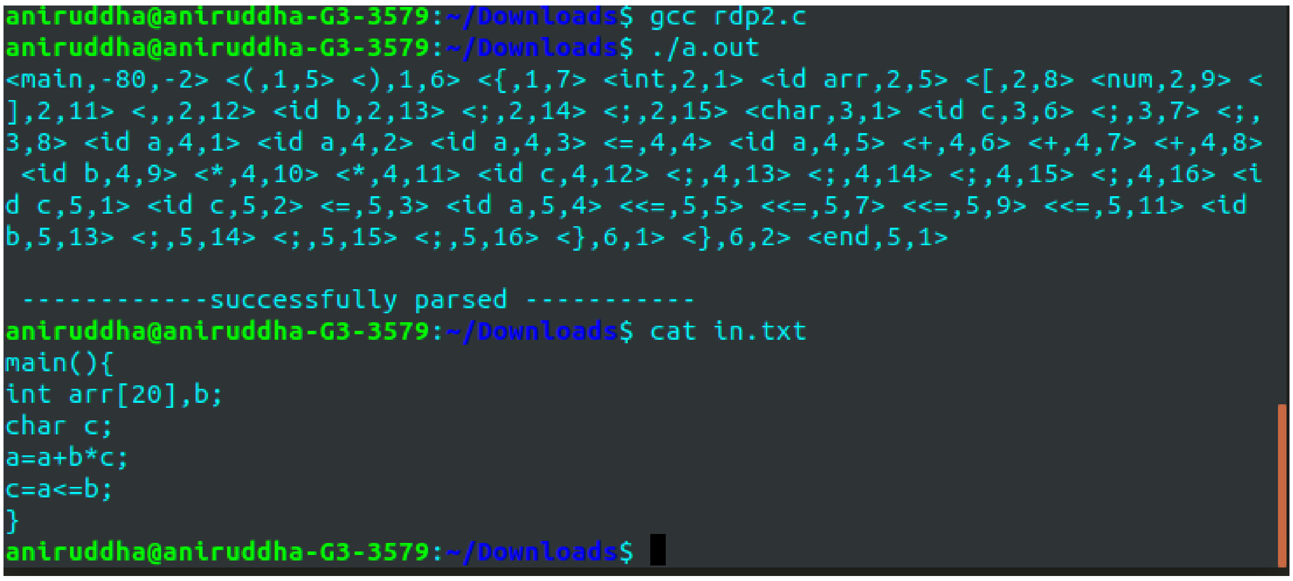
output-

error reporting-



```
aniruddha@aniruddha-G3-3579:~/Downloads$ gcc rdp2.c
aniruddha@aniruddha-G3-3579:~/Downloads$ ./a.out
<main,-80,-2> <(,1,5> <),1,6> <{,1,7> <int,2,1> <id arr,2,5> <[,2,8> <num,2,9> <
],2,11> <,,2,12> <id b,2,13> <;,2,14> <;,2,15> <char,3,1> <id c,3,6> <;,3,7> <;,
3,8> <id a,4,1> <id a,4,2> <id a,4,3> <=,4,4> <id a,4,5> <+,4,6> <+,4,7> <+,4,8>
 <id b,4,9> <*,4,10> <*,4,11> <;,4,12>

error at line:4 and col:13 , expected :identifier or number
aniruddha@aniruddha-G3-3579:~/Downloads$ cat in.txt
main(){
int arr[20],b;
char c;
a=a+b*;
c=a<=b;
}
aniruddha@aniruddha-G3-3579:~/Downloads$
```

successfully parsed-



```
aniruddha@aniruddha-G3-3579:~/Downloads$ gcc rdp2.c
aniruddha@aniruddha-G3-3579:~/Downloads$ ./a.out
<main,-80,-2> <(,1,5> <),1,6> <{,1,7> <int,2,1> <id arr,2,5> <[,2,8> <num,2,9> <
],2,11> <,,2,12> <id b,2,13> <;,2,14> <;,2,15> <char,3,1> <id c,3,6> <;,3,7> <;,
3,8> <id a,4,1> <id a,4,2> <id a,4,3> <=,4,4> <id a,4,5> <+,4,6> <+,4,7> <+,4,8>
 <id b,4,9> <*,4,10> <*,4,11> <id c,4,12> <;,4,13> <;,4,14> <;,4,15> <;,4,16> <i
d c,5,1> <id c,5,2> <=,5,3> <id a,5,4> <<=,5,5> <<=,5,7> <<=,5,9> <<=,5,11> <id
b,5,13> <;,5,14> <;,5,15> <;,5,16> <},6,1> <},6,2> <end,5,1>

 ------------successfully parsed -----------
aniruddha@aniruddha-G3-3579:~/Downloads$ cat in.txt
main(){
int arr[20],b;
char c;
a=a+b*c;
c=a<=b;
}
aniruddha@aniruddha-G3-3579:~/Downloads$
```