**Part A**

**1. a.Program to count the number of characters, words, spaces and lines in a given input file**

**Prog1a.l**

%{

#include<stdio.h>

int c=0,w=0,s=0,l=0;

%}

%%

[^ \t\n,\.:]+ {w++;c=c+yyleng;}

[ ] {s++;}

[\n] {l++;}

. {c++;}

%%

int yywrap()

{

return 1;

}

main(int argc,char \*argv[])

{

if(argc!=2)

{

printf("Usage:<./a.out><sourcefile>\n");

exit(0);

}

yyin=fopen(argv[1],"r");

yylex();

printf("no of character=%d\nNo of words=%d\nNo of spaces=%d\nNo of lines=%d\n",c,w,s,l);

}

**output:**

std.txt

1,abc

2,xyz

3,pqr

4,mno

5 st

lex prog1b.

./a.out std.txt

No. Of character=23

No. Of words=10

No. Of space=1

No. Of line=5

**1.b. Write a LEX program to eliminate comment lines in a C program and copy the resulting program into a separate file.**

**Prog1b.l** -

%{

#include<stdio.h>

int sl=0;

int ml=0;

%}

%%

"/\*"[a-zA-Z0-9' '\t\n]+"\*/" ml++;

"//".\* sl++;

%%

main()

{

yyin=fopen("f1.c","r");

yyout=fopen("f2.c","w");

yylex();

fclose(yyin);

fclose(yyout);

printf("\n Number of single line comments are = %d\n",sl);

printf("\nNumber of multiline comments are =%d\n",ml);

}

**f1.c**

#include<stido.h>

int main()

{

// this is a comment

printf("hello");

/\* this is another comment \*/

}

**Output**

gedit prog1b.l

gedit f1.c

lex prog1b.l

cc lex.yy.c –ll

./a.out

Number of single line comments are =1

Number of multiline comments are =1

**2.a Program to recognize a valid arithmetic expression and to recognize the identifiers and operators present. Print them separately.**

**Prog2a.l**

%{  
#include<stdio.h>  
int v=0,op=0,id=0,flag=0;  
%}  
  
%%  
[a-zA-Z]+[0-9A-Za-z]\* {id++;}  
[0-9]+ {id++;}  
[\+\-\\*/\=] {op++;}  
"(" {v++;}  
")" {v--;}  
";" {flag=1;}  
.|\n {return 0;}  
%%  
  
int main()  
{  
        printf("Enter the expression:");  
        yylex();  
        if((op+1)==id && v==0 && flag==0)  
        {  
                printf("\n Expression is Valid\n");  
                printf("No of identifier = %d \n No of Operators = %d \n",id,op);  
        }  
        else  
                printf("\n Expression is Invalid\n");  
return 0;  
  
}

**Output:**

gedit prog2a.l

lex prog2a.l

cc lex.yy.c –ll

./a.out

Enter the expression: 1+6\*(9-2+4)

Expression is Valid

No of identifier =5

No of Operators =4

**2. b. Program to recognize whether a given sentence is simple or compound.**

prog2b.l

%{

%}

ws[\n\t]+

%%

{ws}”and”{ws} |

{ws}”AND”{ws} |

{ws}”or”{ws} |

{ws}”OR”{ws} |

{ws}”but”{ws} |

{ws}”BUT”{ws} |

{ws}”because”{ws} |

{ws}”neverthles”{ws} {printf(“compound sentence\n”);exit(0);}

.;

\n return 0;

%%

main()

{

Printf(“type the sentence\n”);

yylex();

printf(“simple sentence\n”);

}

**Output:**

Type the sentence:

Hll

Jll

Hello

Simple sentence

**3. Program to recognize and count the number of identifiers in a given input file**

**prog3.l**

%{

#include<stdio.h>

int count=0;

%}

%%

“int” |

“float” |

“chat” |

“double” {char ch;

ch=input();

while(1)

{

if(ch==’,’) count++;

if(ch==’,’) {count++;break;}

//if(ch==’\n’) break;

Ch=input();

}

}

.|\n ;

%%

int main(int argc ,char\*argv[])

{

if(argc!=2)

{

Printf(“\+nUsage:/a.out c.file”);

exit(1);

}

else

{

yyin=fopen(argv[1],”r”);

If(!yyin)

{

Perror(“Opem”);

exit(1);

}

yylex();

printf(“Number of IS=%d\n”,count);

}

return 0;}

**f. c**

chat a[12];

;

;

int 4;

char b;

**Output:**

gedit 3.l

gedit f.c

lex 3.l

cc lex. yy. c -ll

. /a.out f. c

Number of ID=3

**Design, develop, and execute the following programs using YACC:**

**4. a) Program to recognize a valid arithmetic expression that uses operators +, -, \* and /.**

**prog4a.l**

%{

#include”y.tab.h”

%}

%%

[a-z A-Z 0-9]+ {return OPER;}

[\t];

\n {return 0;}

. {return yytext[0];}

%%

**prog4a.y**

%{

#include<stdio..h>

#include<stdlib.h>

%}

%token OPER

&left’+’‘-‘

%left’\*’’/’

%%

expr:e;

e:e’+’e

| e’-’e

| e’\*’e

| e’/’e

| OPER

| ‘(‘e’)’

;

%%

int main()

{  
printf(“Enter the Expression:”);

yyparse();

printf(“Valid Expression:”);

return 0;

}

yyerror()

{

printf(“Invalid Expression\n”);

exit(0);

}

**Output**:

Enter the expression:

2\*5-8

Valid Expression:

**4.b. Program to recognize a valid variable, which starts with a letter, followed by any number of letters ordigits.**

**prog4b.y**

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token L D

%%

S:L X

X:L X|D X|;

%%

Int main()

{

Printf(“Enter the variable:”);

yyparse();

printf(“Valid variable\n”);

return 0;

}

Int yyerror(char \*S)

{

printf(“%s is a Invalid String\n”,S);

exit(1);

}

**prog4b.l**

%{

#include”.y.tab.h”

%}

%%

[a-zA-Z] {return L;}

[0-9] {return D;}

. {yyerror (“SE”);}

[\n] {return 0;}

%%

**Output:**

gedit 4b.l

gedit 4b.y

lex 4b.l

yacc -d 4b.y

cc lex. yy. c y. tab. c -ll

. /a.out

Enter the variable: ABC

Valid variable

Enter the variable: 123

Invalid variable

**5. a) Program to evaluate an arithmetic expression involving operators +, -, \* and /.**

**prog5a.l**

%(

#include”y.tab.h”

Extern int yylval;

%}

%}

[0-9]+ {yylval=atoi(yytext);return NUM;}.

[\t];

[\n] {return 0;}

. {return yytext[0];}

%%

**prog5a.y**

%{

#include<stdio..h>

#include<stdlib.h>

%}

%token NUM

&left’+’‘-‘

%left’\*’’/’

%%

expr:e{printf(“Result=%d\n”,$1);}

e:e’+’e{$$=$1+$3;}

| e’-’e{$$=$-$3;}

| e’\*’e{$$=$1\*$3;}

| e’/’e{$$=$1/$3;}

| ‘(‘e’)’{$$=$2;}

| NUM{$$=$1;}

;

%%

int main()

{  
 printf(“Enter the Expression:”);

yyparse();

printf(“Valid Expression:”);

return 0;

}

yyerror()

{

printf(“Invalid Expression\n”);

exit(1);

}

**Output:**

gedit 5a.l

gedit 5a.y

lex 5a.l

cc lex. yy. c y. tab. c -ll

. /a.out

Enter the expression: 5\*8-3

Result =37

**5.b. Program to recognize strings „aaab‟, „abbb‟, „ab‟ and „a‟ using the grammar (anbn, n>= 0).**

**Prog5b.l**

%{

#include"y.tab.h"

%}

%%

[a] {return A;}

[b] {return B;}

. {yyerror("SE");}

[\n] {return 0;}

%%

**Prog5b.y**

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token A B

%%

S: A S B |;

%%

int main()

{

printf("Enter the string m:");

yyparse();

printf("It is a valid string.");

return 0;

}

int yyerror(char\*S)

{

printf("Invalid String.");

exit(1);

}

**Output:**

gedit prog5a.l

gedit prog5a.y

lex prog5a.l

yacc –d prog5a.y

cc lex.yy.c y.tab.c –ll

./a.out

Enter the string m: aabb

It is a valid string.

Enter the string m: abb

Invalid String

**6. Program to recognize the grammar (an b, n>= 10).**

**Prog6.l**

%{

#include "y.tab.h"

%}

%%

[a] {return A;}

[b] {return B;}

. {yyerror("SE");}

[\n] {return 0;}

%%

**Prog6.y**

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token A B

%%

S:A A A A A A A A A A X B;

X:A X |;

%%

int main()

{

printf("Enter the valid string:");

yyparse();

printf("Valid string:");

return 0;

}

int yyerror(char \*S)

{

printf("Invalid String.");

exit(1);

}

**Output:**

Enter the valid string: aaaaaaaaaab

Valid string

Enter the valid string: aaaaab

Invalid String.

**Part B**

**7. -Design, develop and implement YACC/C program to construct Predictive / LL(1) Parsing Table for the grammar rules: A ->aBa , B ->bB | e. Use this table to parse the sentence: abba$**

**Lab7.c** - **PROGRAM**

/\*GRAMMER RULES ---- A ->aBa , B ->bB | @\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char prod [3][10]={"A->aBa","B->bB","B->@"};

char first[3][10]={"a","b","@"};

char follow[3][10]={"$","a","a"};

char table[3][4][10];

char input[10];

int top=-1;

char stack[25];

char curp[20];

void push(char item)

{

  stack[++top]=item;

}

void pop()

{

  top=top-1;

}

void display()

{

  int i;

  for(i=top;i>=0;i--)

  printf("%c",stack[i]);

}

int numr(char c)

{

  switch(c)

  {

    case'A':return 1;

    case'B':return 2;

    case'a':return 1;

    case'b':return 2;

    case'@':return 3;

  }

  return 1;

}

int main()

{

  char c;

  int i,j,k,n;

  for(i=0;i<3;i++){

    for(j=0;j<4;j++){

      strcpy(table[i][j],"EMPTY");

    }

  }

  printf("\nGrammar\n");

  for(i=0;i<3;i++)

  printf("%s\n",prod[i]);

  printf("\nfirst={%s,%s,%s}",first[0],first[1],first[2]);

  printf("\nfollow={%s,%s}\n",follow[0],follow[1]);

  printf("\nPredictive parsing table for the given grammar :\n");

  strcpy(table[0][0],"");

  strcpy(table[0][1],"a");

  strcpy(table[0][2],"b");

  strcpy(table[0][3],"$");

  strcpy(table[1][0],"A");

  strcpy(table[2][0],"B");

  for(i=0;i<3;i++)

  {

    if(first[i][0]!='@')

    strcpy(table[numr(prod[i][0])][numr(first[i][0])],prod[i]);

    else

    strcpy(table[numr(prod[i][0])][numr(follow[i][0])],prod[i]);

  }

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

  for(i=0;i<3;i++){

    for(j=0;j<4;j++)

    {

      printf("%-30s",table[i][j]);

      if(j==3) printf("\n-------------------------------------------------------------------\n");

    }

  }

  printf("Enter the input string terminated with $ to parse:-");

  scanf("%s",input);

  for(i=0;input[i]!='\0';i++){

    if((input[i]!='a')&&(input[i]!='b')&&(input[i]!='$'))

    {

      printf("Invalid String");

      exit(0);

    }

  }

if(input[i-1]!='$')

  {

    printf("\n\nInput String Entered Without End Marker $");

    exit(0);

  }

push('$');

  push('A');

  i=0;

printf("\n\n");

  printf("Stack\t Input\tAction");

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

while(input[i]!='$'&&stack[top]!='$')

  {

    display();

    printf("\t\t%s\t",(input+i));

    if(stack[top]==input[i])

    {

      printf("\tMatched %c\n", input[i]);

      pop();

      i++;

    }

    else

    {

      if(stack[top]>=65&&stack[top]<92)

      {

        strcpy(curp,table[numr(stack[top])][numr(input[i])]);

        if(!(strcmp(curp,"e")))

        {

          printf("\nInvalid String - Rejected\n");

          exit(0);

        }

        else

        {

          printf("\tApply production %s\n",curp);

          if(curp[3]=='@')

          pop();

          else

          {

            pop();

            n=strlen(curp);

            for(j=n-1;j>=3;j--)

            push(curp[j]);

          }

        }

      }

    }

  }

  display();

  printf("\t\t%s\t",(input+i));

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

  if(stack[top]=='$'&&input[i]=='$')

  {

    printf("\nValid String - Accepted\n");

  }

  else

  {

    printf("Invalid String - Rejected\n");

  }

}

[Text

Description automatically generated](https://4.bp.blogspot.com/-_A4KiyEp19I/XM1zdTJ2F6I/AAAAAAAAIuY/85TLop8VYfUjsBJ_j9tnUoyXarR5jBaAACEwYBhgL/s1600/OUTPUT3.JPG)

**8. Design, develop and implement YACC/C program to demonstrate Shift Reduce Parsing technique for the grammar rules: E ®E+T | T, T ®T\*F | F, F ®(E) | id and parse the sentence: id + id \* id.**  
 **lab8.c - PROGRAM**  
  
#include<stdio.h>  
#include<string.h>  
  
int k=0,z=0,i=0,j=0,c=0;  
char a[16],ac[20],stk[15],act[10];  
void check();  
  
int main()  
{  
    puts("GRAMMAR is E->E+E \n E->E\*E \n E->(E) \n E->id");  
    puts("\nEnter input string :");  
    gets(a);  
    c=strlen(a);  
    strcpy(act,"SHIFT->");  
    puts("stack \t input \t action");  
    for(k=0,i=0; j<c; k++,i++,j++)  
    {  
        if(a[j]=='i' && a[j+1]=='d')  
        {  
            stk[i]=a[j];  
            stk[i+1]=a[j+1];  
            stk[i+2]='\0';  
            a[j]=' ';  
            a[j+1]=' ';  
            printf("\n$%s\t%s$\t%sid",stk,a,act);  
            check();  
        }  
        else  
        {  
            stk[i]=a[j];  
            stk[i+1]='\0';  
            a[j]=' ';  
            printf("\n$%s\t%s$\t%ssymbols",stk,a,act);check();  
        }  
    }  
}  
  
void check()  
{  
    strcpy(ac,"REDUCE TO E");  
    for(z=0; z<c; z++)  
        if(stk[z]=='i' && stk[z+1]=='d')  
        {  
            stk[z]='E';  
            stk[z+1]='\0';  
            printf("\n$%s\t%s$\t%s",stk,a,ac);  
            j++;  
        }  
    for(z=0; z<c; z++)  
        if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')  
        {  
            stk[z]='E';  
            stk[z+1]='\0';  
            stk[z+2]='\0';  
            printf("\n$%s\t%s$\t%s",stk,a,ac);  
            i=i-2;  
        }  
    for(z=0; z<c; z++)  
        if(stk[z]=='E' && stk[z+1]=='\*' && stk[z+2]=='E')  
        {  
            stk[z]='E';  
            stk[z+1]='\0';  
            stk[z+2]='\0';  
            printf("\n$%s\t%s$\t%s",stk,a,ac);  
            i=i-2;  
        }  
    for(z=0; z<c; z++)  
        if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')  
        {  
            stk[z]='E';  
            stk[z+1]='\0';  
            stk[z+1]='\0';  
            printf("\n$%s\t%s$\t%s",stk,a,ac);  
            i=i-2;  
        }  
  
}

**OUTPUT :**

[Text

Description automatically generated](https://1.bp.blogspot.com/-iAYW-bMSrHM/XM1zeYilakI/AAAAAAAAIuU/s5LNMiP65TIXoBYjneBfwSw4dSmdCL8uACEwYBhgL/s1600/OUTPUT4.JPG)

**9. Design, develop and implement the syntax-directed definition of “if E then S1” and “if E then S1 else S2”**

#include<stdio.h>

#include<stdlib.h>

int parsecondition(char[ ],int,char\*,int);

void gen(char[ ],char[ ],char[ ],int);

int main()

{

int counter=0,stlen=0,elseflag=0;

char stmt[60];

char strB[54];

char strS1[50];

char strS2[45];

printf("Format of 'if' statement\n example..\n");

printf("if(a”);

printf("if(a\n”);

printf("Enter the statement\n");

scanf("%[^\n]",stmt);

stlen=strlen(stmt);

counter=counter+2;

counter=parsecondition(stmt,counter,strB,stlen);

if(stmt[counter]==')')

counter++;

counter=counter+3;

counter=parsecondition(stmt,counter,strS1,stlen);

if(stmt[counter+1]==';')

{

printf("\n parsing the input statement");

gen(strB,strS1,strS2,elseflag);

return 0;

}

if(stmt[counter]==')')

counter++;

counter=counter+3;

counter=parsecondition(stmt,counter,strS2,stlen);

counter=counter+2;

if(counter==stlen)

{

elseflag=1;

printf("\n parsing the input statement");

gen(strB,strS1,strS2,elseflag);

return 0;

}

return 0;

}

int parsecondition(char input[ ],int cntr,char\* dest,int totallen)

{

int index=0,pos=0;

while(input[cntr]!='(' && cntr<=totallen)

cntr++;

if(cntr>=totallen)

return 0;

index=cntr;

while(input[cntr]!=')')

cntr++;

if(cntr>=totallen)

return 0;

while(index<=cntr)

dest[pos++]=input[index++];

dest[pos]='\0';

return cntr;

}

void gen(char B[ ],char S1[ ],char S2[ ],int elsepart)

{

int Bt=101,Bf=102,Sn=103;

printf("\n\t if %s goto%d",B,Bt);

printf("\n\tgoto %d",Bf);

printf("\n %d:",Bt);

printf("%s",S1);

if(!elsepart)

printf("\n%d\n",Bf);

else

{

printf("\n\t goto %d",Sn);

printf("\n %d:%s",Bf,S2);

printf("\n%d\n",Sn);

}

}

**Output:**

Format of 'if' statement

example..

if(a

if(a

Enter the statement

if(a

parsing the input statement

if (a

goto 102

101:(x=a)

102

**10. Write a yacc program that accepts a regular expression as input and produce its parse tree as output.**

%{/\*declaration part\*/

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#include<string.h>

#define MAX 100 /\*to store productions\*/

int getREindex ( const char\* );

signed char productions[MAX][MAX];

int count = 0 , i , j;

char temp[200] , temp2[200];

%}

%token ALPHABET

%left '|'

%left '.'

%nonassoc '\*' '+'

%%/\*rules section\*/

S : re '\n' {

printf ( "This is the rightmost derivation--\n" );

for ( i = count - 1 ; i >= 0 ; --i ) {

if ( i == count - 1 ) {

printf ( "\nre => " );

strcpy ( temp , productions[i] );

printf ( "%s" , productions[i] );

}

else {

printf ( "\n => " );

j = getREindex ( temp );

temp[j] = '\0';

sprintf ( temp2 , "%s%s%s" , temp , productions[i] , (temp + j + 2) );

printf ( "%s" , temp2 );

strcpy ( temp , temp2 );

}

}

printf ( "\n" );

exit ( 0 );

}

re : ALPHABET {

temp[0] = yylval; temp[1] = '\0';

strcpy ( productions[count++] , temp );/\*copy the input to the prodcution array\*/

}/\*only conditions defined here will be valid, this is the structure\*/

| '(' re ')' /\*adds the (expression) to the production array\*/

{ strcpy ( productions[count++] , "(re)" ); }

| re '\*'

{ strcpy ( productions[count++] , "re\*" ); }

| re '+' /\*adds expression+ type to the production array\*/

{ strcpy ( productions[count++] , "re+" ); }

| re '|' re /\*adds the expression|expression to the production array\*/

{strcpy ( productions[count++] , "re | re" );}

| re '.' re/\*adds the expression.expression to the production array\*/

{strcpy ( productions[count++] , "re . re" );}

;

%%

int main ( int argc , char \*\*argv )

{

/\*

Parse and output the rightmost derivation,

from which we can get the parse tree

\*/

yyparse();/\*calls the parser\*/

return 0;

}

yylex() /\*calls lex and takes each character as input and feeds ALPHABET to check for the structure\*/

{

signed char ch = getchar();

yylval = ch;

if ( isalpha ( ch ) )

return ALPHABET;

return ch;

}

yyerror() /\*Function to alert user of invalid regular expressions\*/

{

fprintf(stderr , "Invalid Regular Expression!!\n");

exit ( 1 );

}

int getREindex ( const char \*str )

{

int i = strlen ( str ) - 1;

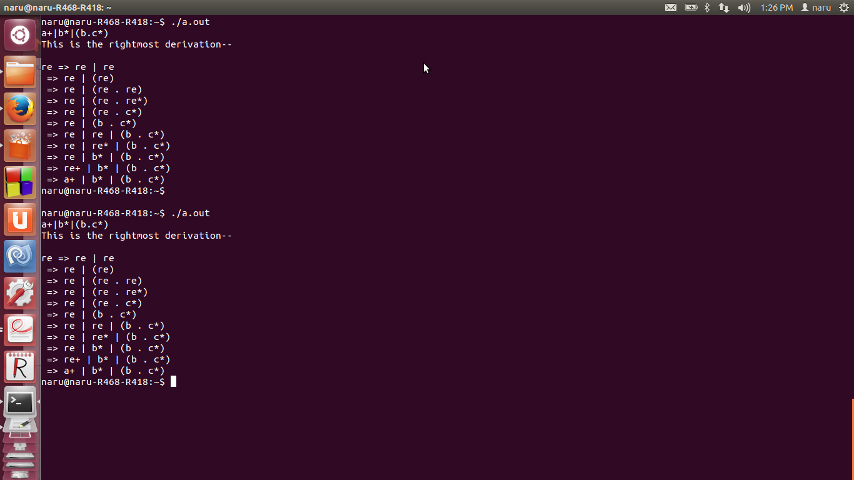
for ( ; i >= 0 ; --i ) {

if ( str[i] == 'e' && str[i-1] == 'r' )

return i-1;

}

}



**11- Design, develop and implement a C/Java program to generate the machine code using**  
**Triples for the statement A = -B \* (C +D) whose intermediate code in three-address form:**  
 **T1 = -B**  
**T2 = C + D**  
**T3 = T1 + T2**  
**A = T3**  
 **la11.c** - **PROGRAM**  
#include<stdio.h>  
#include<stdlib.h>  
#include<ctype.h>  
  
char op[2],arg1[5],arg2[5],result[5];  
int main()  
{  
    FILE \*fp1,\*fp2;  
    fp1=fopen("input.txt","r");  
    fp2=fopen("output.txt","w");  
    while(!feof(fp1))  
    {  
        fscanf(fp1,"%s%s%s%s",result,arg1,op,arg2);  
        if(strcmp(op,"+")==0)  
        {  
            fprintf(fp2,"\nMOV R0,%s",arg1);  
            fprintf(fp2,"\nADD R0,%s",arg2);  
            fprintf(fp2,"\nMOV %s,R0",result);  
        }  
  
        if(strcmp(op,"\*")==0)  
        {  
            fprintf(fp2,"\nMOV R0,%s",arg1);  
            fprintf(fp2,"\nMUL R0,%s",arg2);  
            fprintf(fp2,"\nMOV %s,R0",result);  
        }  
  
        if(strcmp(op,"-")==0)  
        {  
            fprintf(fp2,"\nMOV R0,%s",arg1);  
            fprintf(fp2,"\nSUB R0,%s",arg2);  
            fprintf(fp2,"\nMOV %s,R0",result);  
        }  
  
        if(strcmp(op,"/")==0)  
        {  
            fprintf(fp2,"\nMOV R0,%s",arg1);  
            fprintf(fp2,"\nDIV R0,%s",arg2);  
            fprintf(fp2,"\nMOV %s,R0",result);  
        }  
  
        if(strcmp(op,"=")==0)  
        {  
            fprintf(fp2,"\nMOV R0,%s",arg1);  
            fprintf(fp2,"\nMOV %s,R0",result);  
        }  
    }  
    fclose(fp1);  
    fclose(fp2);  
}

**input.txt**

T1 -B = ?

T2 C + D

T3 T1 \* T2

A T3 = ?

**OUTPUT :**( click on image to zoom )

[A screenshot of a computer

Description automatically generated with medium confidence](https://1.bp.blogspot.com/-loMUrJQvS7A/XM1zegEd0HI/AAAAAAAAIuc/Rf6NsNh0Zkg-rsD82G44UoehJqPilZkbQCEwYBhgL/s1600/OUTPUT5.JPG)