```
#include<stdio.h>
#include<stdlib.h>
struct node {
    int st;
    struct node *link;
};
struct node1 {
    int nst[20];
};

void insert(int ,char, int);
int findalpha(char);
void findfinalstate(void);
int insertdfastate(struct node1);
int compare(struct node1,struct node1);
void printnewstate(struct node1);
static int
set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,buffer[20];
int complete=-1;
char alphabet[20];
static int eclosure[20][20]={0};
struct node1 hash[20];
struct node * transition[20][20]={NULL};
void main() {
    int i,j,k,m,t,n,l;
    struct node *temp;
    struct node1 newstate={0},tmpstate={0};

    printf("Enter the number of alphabets?\n");
    printf("NOTE:- [ use letter e as epsilon]\n");
    printf("NOTE:- [e must be last character ,if it is present]\n");
    printf("\nEnter No of alphabets and alphabets?\n");
    scanf("%d",&noalpha);
    getchar();
    for(i=0;i<noalpha;i++) {
        alphabet[i]=getchar();
        getchar();
    }
    printf("Enter the number of states?\n");
    scanf("%d",&nostate);
    printf("Enter the start state?\n");
    scanf("%d",&start);
    printf("Enter the number of final states?\n");
    scanf("%d",&nofinal);
    printf("Enter the final states?\n");
    for(i=0;i<nofinal;i++)
    scanf("%d",&finalstate[i]);
    printf("Enter no of transition?\n");

    scanf("%d",&notransition);
    printf("NOTE:- [Transition is in the form-> qno alphabet qno]\n",notransition);
    printf("NOTE:- [States number must be greater than zero]\n");
    printf("\nEnter transition?\n");

    for(i=0;i<notransition;i++) {
        scanf("%d %c%d",&r,&c,&s);
        insert(r,c,s);
    }
    for(i=0;i<20;i++) {
        for(j=0;j<20;j++) hash[i].nst[j]=0;
    }
    complete=-1;
    i=-1;
    printf("\nEquivalent DFA.....\n");
    printf(".....\n");
    printf("Trnsitions of DFA\n");
    newstate.nst[start]=start;
    insertdfastate(newstate);
    while(i!=complete) {
        i++;
        newstate=hash[i];
        for(k=0;k<noalpha;k++) {
            c=0;
            for(j=1;j<=nostate;j++) set[j]=0;
            for(j=1;j<=nostate;j++) {
                l=newstate.nst[j];
                if(l!=0) {
                    temp=transition[l][k];
                    while(temp!=NULL) {
                        if(set[temp->st]==0) {
                            c++;
                            set[temp->st]=temp->st;
                        }
                        temp=temp->link;
                    }
                }
            }
            printf("\n");
            if(c!=0) {
                for(m=1;m<=nostate;m++)
                    tmpstate.nst[m]=set[m];

                insertdfastate(tmpstate);
                printnewstate(newstate);
                printf("%c\t",alphabet[k]);
                printnewstate(tmpstate);
                printf("\n");
            } else {
                printnewstate(newstate);
                printf("%c\t", alphabet[k]);
                printf("NULL\n");
            }
        }
    }
    printf("\nStates of DFA:\n");
    for(i=0;i<=complete;i++)
        printnewstate(hash[i]);
    printf("\n Alphabets:\n");
    for(i=0;i<noalpha;i++)
        printf("%c\t",alphabet[i]);
    printf("\n Start State:\n");
    printf("q%d",start);
    printf("\nFinal states:\n");
    findfinalstate();
}
int insertdfastate(struct node1 newstate) {
    int i;
    for(i=0;i<=complete;i++) {
```

```
        if(compare(hash[i],newstate))
            return 0;
    }
    complete++;
    hash[complete]=newstate;
    return 1;
}

int compare(struct node1 a,struct node1 b) {
    int i;
    for(i=1;i<=nostate;i++) {
        if(a.nst[i]!=b.nst[i])
            return 0;
    }
    return 1;
}

void insert(int r,char c,int s) {
    int j;
    struct node *temp;
    j=findalpha(c);
    if(j==999) {
        printf("error\n");
        exit(0);
    }
    temp=(struct node *) malloc(sizeof(struct node));
    temp->st=s;
    temp->link=transition[r][j];
    transition[r][j]=temp;
}

int findalpha(char c) {
    int i;
    for(i=0;i<noalpha;i++)
    if(alphabet[i]==c)
        return i;

    return(999);
}


void findfinalstate() {
    int i,j,k,t;

    for(i=0;i<=complete;i++) {
        for(j=1;j<=nostate;j++) {
            for(k=0;k<nofinal;k++) {
                if(hash[i].nst[j]==finalstate[k]) {
                    printnewstate(hash[i]);
                    printf("\t");
                    j=nostate;
                    break;
                }
            }
        }
    }
}
```

```
void printnewstate(struct node1 state) {
    int j;
    printf("{");
    for(j=1;j<=nostate;j++) {
    if(state.nst[j]!=0)
        printf("q%d,",state.nst[j]);
    }
    printf("}\t");
}
```

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int st;
    struct node *link;
};

void findclosure(int,int);
void insert_trantbl(int ,char, int);
int findalpha(char);
void findfinalstate(void);
void unionclosure(int);
void print_e_closure(int);
static int
set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,buffer[20];
char alphabet[20];
static int e_closure[20][20]={0};
struct node * transition[20][20]={NULL};

void main(){
    int i,j,k,m,t,n;

    struct node *temp;
    printf("enter the number of alphabets?\n");
    scanf("%d",&noalpha);
    getchar();
    printf("NOTE:- [ use letter e as epsilon]\n");
    printf("NOTE:- [e must be last character ,if it is present]\n");

    printf("\nEnter alphabets?\n");
    for(i=0;i<noalpha;i++){
        alphabet[i]=getchar();
        getchar();
    }
    printf("Enter the number of states?\n");
    scanf("%d",&nostate);
    printf("Enter the start state?\n");
    scanf("%d",&start);
    printf("Enter the number of final states?\n");
    scanf("%d",&nofinal);
    printf("Enter the final states?\n");
    for(i=0;i<nofinal;i++)
            scanf("%d",&finalstate[i]);
    printf("Enter no of transition?\n");
    scanf("%d",&notransition);
    printf("NOTE:- [Transition is in the form--> qno   alphabet   qno]\
n",notransition);
    printf("NOTE:- [States number must be greater than zero]\n");
    printf("\nEnter transition?\n");
    for(i=0;i<notransition;i++) {
            scanf("%d %c%d",&r,&c,&s);
            insert_trantbl(r,c,s);
    }
    printf("\n");

    for(i=1;i<=nostate;i++) {
            c=0;
            for(j=0;j<20;j++) {
                buffer[j]=0;
                e_closure[i][j]=0;
            }
            findclosure(i,i);
    }
    printf("Equivalent NFA without epsilon\n");
    printf("-----------------------------------\n");
    printf("start state:");
    print_e_closure(start);
    printf("\nAlphabets:");
    for(i=0;i<noalpha;i++)
        printf("%c ",alphabet[i]);
    printf("\n States :" );
    for(i=1;i<=nostate;i++)
        print_e_closure(i);
    printf("\nTnransitions are...:\n");
    for(i=1;i<=nostate;i++) {
        for(j=0;j<noalpha-1;j++) {
            for(m=1;m<=nostate;m++) set[m]=0;
            for(k=0;e_closure[i][k]!=0;k++) {
                t=e_closure[i][k];
                temp=transition[t][j];
                while(temp!=NULL) {
                    unionclosure(temp->st);
                    temp=temp->link;
                }
            }
            printf("\n");
            print_e_closure(i);
            printf("%c\t",alphabet[j]   );
            printf("{");
            for(n=1;n<=nostate;n++) {
                if(set[n]!=0)
                        printf("q%d,",n);
            }
            printf("}");
        }
    }
    printf("\n Final states:");
    findfinalstate();
}

void findclosure(int x,int sta) {
    struct node *temp;
    int i;
    if(buffer[x])
        return;
    e_closure[sta][c++]=x;
    buffer[x]=1;
    if(alphabet[noalpha-1]=='e' && transition[x][noalpha-1]!=NULL) {
        temp=transition[x][noalpha-1];
        while(temp!=NULL) {
            findclosure(temp->st,sta);
            temp=temp->link;
        }
    }
}

void insert_trantbl(int r,char c,int s) {
```

```
    int j;
    struct node *temp;
    j=findalpha(c);
    if(j==999) {
        printf("error\n");
        exit(0);
    }
    temp=(struct node *) malloc(sizeof(struct node));
    temp->st=s;
    temp->link=transition[r][j];
    transition[r][j]=temp;
}

int findalpha(char c) {
    int i;
    for(i=0;i<noalpha;i++)
            if(alphabet[i]==c)
                    return i;
    return(999);
}

void unionclosure(int i) {
    int j=0,k;
    while(e_closure[i][j]!=0) {
        k = e_closure[i][j];
        set[k]=1;
        j++;
    }
}
void findfinalstate() {
    int i,j,k,t;
    for(i=0;i<nofinal;i++) {
        for(j=1;j<=nostate;j++) {
            for(k=0;e_closure[j][k]!=0;k++) {
                if(e_closure[j][k]==finalstate[i]) {
                    print_e_closure(j);
                }
            }
        }
    }
}

void print_e_closure(int i) {
    int j;
    printf("{");
    for(j=0;e_closure[i][j]!=0;j++)
        printf("q%d,",e_closure[i][j]);
    printf("}\t");
}
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

char *input;
int i=0;
char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E","i","E^E"};

int top=0,l;
char prec[9][9]={

                            /*input*/

            /*stack    +    -   *   /   ^   i   (   )   $  */

            /*  + */  '>', '>','<','<','<','<','<','>','>',

            /*  - */  '>', '>','<','<','<','<','<','>','>',

            /*  * */  '>', '>','>','>','<','<','<','>','>',

            /*  / */  '>', '>','>','>','<','<','<','>','>',

            /*  ^ */  '>', '>','>','>','<','<','<','>','>',

            /*  i */  '>', '>','>','>','>','e','e','>','>',

            /*  ( */  '<', '<','<','<','<','<','<','>','e',

            /*  ) */  '>', '>','>','>','>','e','e','>','>',

            /*  $ */  '<', '<','<','<','<','<','<','<','>',

                };

int getindex(char c)
{
switch(c)
    {
    case '+':return 0;
    case '-':return 1;
    case '*':return 2;
    case '/':return 3;
    case '^':return 4;
    case 'i':return 5;
    case '(':return 6;
    case ')':return 7;
    case '$':return 8;
    }
}


int shift()
{
stack[++top]=*(input+i++);
stack[top+1]='\0';
}
```

```
int reduce()
{
int i,len,found,t;
for(i=0;i<5;i++)
    {
    len=strlen(handles[i]);
    if(stack[top]==handles[i][0]&&top+1>=len)
        {
        found=1;
        for(t=0;t<len;t++)
            {
            if(stack[top-t]!=handles[i][t])
                {
                found=0;
                break;
                }
            }
        if(found==1)
            {
            stack[top-t+1]='E';
            top=top-t+1;
            strcpy(lasthandle,handles[i]);
            stack[top+1]='\0';
            return 1;
            }
        }
   }
return 0;
}



void dispstack()
{
int j;
for(j=0;j<=top;j++)
    printf("%c",stack[j]);
}



void dispinput()
{
int j;
for(j=i;j<l;j++)
    printf("%c",*(input+j));
}



void main()
{
int j;

input=(char*)malloc(50*sizeof(char));
printf("\nEnter the string\n");
scanf("%s",input);
input=strcat(input,"$");
l=strlen(input);

strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
      {
      shift();
      printf("\n");
      dispstack();
      printf("\t");
      dispinput();
      printf("\tShift");
      if(prec[getindex(stack[top])][getindex(input[i])]=='>')
            {
            while(reduce())
                  {
                  printf("\n");
                  dispstack();
                  printf("\t");
                  dispinput();
                  printf("\tReduced: E->%s",lasthandle);
                  }
            }
      }

if(strcmp(stack,"$E$")==0)
    printf("\nAccepted\n");
else
    printf("\nNot Accepted\n");
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int **transitionMap;
int **partitionTransitionMap;
int startState;
long int reachable;
long int allStates;
long int finalStates;
long int nonFinalStates;
long int *P;

void dfs(int v)
{
      reachable |= (1 << v);
      for(int i=0; i<26; i++)
            if((transitionMap[v][i] != -1) && ((reachable & (1 << transitionMap[v]
[i])) == 0))
                  dfs(transitionMap[v][i]);
}


int main(){
      finalStates = 0;
      allStates = 0;
      transitionMap = (int**)malloc(64*sizeof(int*));
      for (int i = 0; i < 64; i++){
            transitionMap[i] = (int*) malloc(26*sizeof(int));
            for (int j = 0; j < 26; j++){
                  transitionMap[i][j] = -1;
            }
      }

      partitionTransitionMap = (int**)malloc(64*sizeof(int*));
      for (int i = 0; i < 64; i++){
            partitionTransitionMap[i] = (int*) malloc(26*sizeof(int));
            for (int j = 0; j < 26; j++){
                  partitionTransitionMap[i][j] = -1;
            }
      }
      char buff[125];
      printf("\nEnter the start state\n");
      fgets(buff, sizeof(buff), stdin);
      char *p = strtok(buff, " ");
      startState = atoi(p);
      printf("\nEnter the final state(s)\n");
      fgets(buff, sizeof(buff), stdin);
      p = strtok(buff, " ");
      while (p != NULL)
      {
            int state = atoi(p);
            finalStates |= 1 << (state);
            p = strtok(NULL, " ");
      }
      int from;
      char symbol;
      int to;
      printf("\nEnter the transitions one by one in the form state symbol state.\n
Press Ctrl+D when finished\n");
      while (fscanf(stdin, "%d %c %d", &from, &symbol, &to) != EOF) {
            transitionMap[from][symbol-'a'] = to;
            allStates |= (1 << from);
            allStates |= (1 << to);
      }
      reachable = 0;
      dfs(startState);
      allStates &= reachable;
      finalStates &= reachable;
      P =  (long int*) malloc(64*sizeof(long int));
      for (int i = 0; i < 64 ; i++){
            P[i] = 0;
      }
      nonFinalStates = allStates & ~finalStates;
      P[0] = finalStates;
      P[1] = nonFinalStates;
      int nextPartitionIndex = 2;
      for (int i = 0; i < 64; i++){
            long int newPartition = 0;
            if (P[i] == 0)
                  break;
            for (int j = 63; j >=  0; j--) {
                  long int staticState = (long int) 1 << j;
                  if ((P[i] & (staticState)) != 0){
                        partitionTransitionMap[i] = transitionMap[j];
                        for (int k = j - 1; k >= 0; k -- ){
                              long int otherState = (long int) 1 << k;
                              if ((P[i] & (otherState)) != 0){
                                    for (int l  = 0; l < 26; l++){
                                          int staticNext = -1;
                                          int otherNext = -1;
                                          for (int m = 0; m < nextPartitionIndex;
m++){
                                                if ((P[m] & (1 << transitionMap[j]
[l])) != 0)
                                                      staticNext = m;
                                                if ((P[m] & (1 << transitionMap[k]
[l])) != 0)
                                                      otherNext = m;
                                          }
                                          if (transitionMap[j][l] !=
transitionMap[k][l] && (staticNext != otherNext)){
                                                P[i] &= ~(1 << k);
                                                newPartition |= (1 << k);
                                                break;
                                          }
                                    }
                              }
                        }
                        break;
                  }
            }
            if (newPartition != 0){
                  P[nextPartitionIndex] = newPartition;
                  nextPartitionIndex++;
            }
      }
      int startPartition = 0;
```

```
      for (int i = 0; i < nextPartitionIndex; i ++){
            if ((P[i] & (1 << startState)) != 0 ){
                  startPartition = i;
                  break;
            }
      }
      printf("\nThe new start state is:\n");
      printf("%d \n", startPartition);
      printf("\nthe new final state(s) is/are:\n");
      for (int i = 0; i < nextPartitionIndex; i++){
            if ((P[i] & finalStates) != 0){
                  printf("%d ", i);
            }
      }
      printf("\n");
      printf("\nThe new transitions are:\n");
      for (int i = 0; i < nextPartitionIndex; i++){
            for (int j = 0; j < 26; j++) {
                  if (partitionTransitionMap[i][j] != -1){
                        for (int k = 0; k < nextPartitionIndex; k++){
                              if ((P[k] & (1 << partitionTransitionMap[i][j])) !=
0){
                                    printf("%d %c %d\n", i, j + 'a', k);
                              }
                        }
                  }
            }
      }

      return 0;

}
```

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>

char input[10];
int i, error;
void E();
void T();
void Eprime();
void Tprime();
void F();
void main() {
    i = 0;
    error = 0;
    printf("Enter an arithmetic expression : ");
    gets(input);
    E();
    if (strlen(input) == i && error == 0)
        printf("\nAccepted..!!!\n");
    else
        printf("\nRejected..!!!\n");
}

void E(){
    T();
    Eprime();
}
void Eprime() {
  if (input[i] == '+') {
    i++;
    T();
    Eprime();
  }
}
void T() {
  F();
  Tprime();
}
void Tprime() {
  if (input[i] == '*') {
    i++;
    F();
    Tprime();
  }
}
void F() {
  if (isalnum(input[i])) i++;
  else if (input[i] == '(') {
    i++;
    E();
    if (input[i] == ')') i++;
    else error = 1;
  } else
    error = 1;
}
```

```c
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>

int n, m = 0, p, i = 0, j = 0;
char a[10][10], f[10];
void follow(char c);
void first(char c);
int main() {
    int i, z;
    char c, ch;
    printf("Enter the no of productions : \n");
    scanf("%d", &n);
    printf("Enter the productions:\n");
    for (i = 0; i < n; i++)
        scanf("%s%c", a[i], & ch);
    do {
        m = 0;
        printf("Enter a variable whose fisrt & follow is to be found:");
        scanf("%c", & c);
        first(c);
        printf("First(%c)={", c);
        for (i = 0; i < m; i++) printf("%c", f[i]);
        printf("}\n");
        strcpy(f, " ");
        m = 0;
        follow(c);
        printf("Follow(%c)={", c);
        for (i = 0; i < m; i++) printf("%c", f[i]);
        printf("}\n");
        printf("Want to continue or not(1/0) ? ");
        scanf("%d%c", & z, & ch);
  } while (z == 1);
  return (0);
}

void first(char c) {
    int k;
    if (!isupper(c)) f[m++] = c;
    for (k = 0; k < n; k++) {
        if (a[k][0] == c) {
            if (a[k][2] == '$')
                follow(a[k][0]);
            else if (islower(a[k][2]))
                f[m++] = a[k][2];
            else
                first(a[k][2]);
        }
    }
}

void follow(char c) {
    if (a[0][0] == c)
        f[m++] = '$';
    for (i = 0; i < n; i++) {
        for (j = 2; j < strlen(a[i]); j++) {
            if (a[i][j] == c) {
                if (a[i][j + 1] != '\0')
                    first(a[i][j + 1]);
                if (a[i][j + 1] == '\0' && c != a[i][0])
                    follow(a[i][0]);

            }
        }
    }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int i = 1, j = 0, no = 0, tmpch = 90;
char str[100], left[15], right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp {
    int pos;
    char op;
} k[15];

void main(){
    printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
    printf("Enter the Expression :");
    scanf("%s", str);
    printf("The intermediate code:\n");
    findopr();
    explore();
}

void findopr(){
    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == ':') {
            k[j].pos = i;
            k[j++].op = ':';
        }
    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == '/') {
            k[j].pos = i;
            k[j++].op = '/';
        }
    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == '*') {
            k[j].pos = i;
            k[j++].op = '*';
        }
    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == '+') {
            k[j].pos = i;
            k[j++].op = '+';
        }
    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == '-'){
            k[j].pos = i;
            k[j++].op = '-';
        }
}

void explore(){
    i = 1;
    while (k[i].op != '\0') {
        fleft(k[i].pos);
        fright(k[i].pos);
        str[k[i].pos] = tmpch--;
        printf("\t%c := %s%c%s\t\t", str[k[i].pos], left, k[i].op, right);
        printf("\n");
        i++;
    }
    fright(-1);
    if (no == 0) {
        fleft(strlen(str));
        printf("\t%s := %s", right, left);
        exit(0);
    }
    printf("\t%s := %c", right, str[k[--i].pos]);
}

void fleft(int x) {
    int w = 0, flag = 0;
    x--;
    while (x != -1 && str[x] != '+' && str[x] != '*' && str[x] != '=' && str[x] !=
'\0' && str[x] != '-' && str[x] != '/' && str[x] != ':'){
        if (str[x] != '$' && flag == 0){
            left[w++] = str[x];
            left[w] = '\0';
            str[x] = '$';
            flag = 1;
        }
        x--;
    }
}

void fright(int x) {
    int w = 0, flag = 0;
    x++;
    while (x != -1 && str[x] != '+' && str[x] != '*' && str[x] != '\0' && str[x] !=
'=' && str[x] != ':' && str[x] != '-' && str[x] != '/') {
        if (str[x] != '$' && flag == 0){
            right[w++] = str[x];
            right[w] = '\0';
            str[x] = '$';
            flag = 1;
        }
        x++;
    }
}
```

```
#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("enter 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 + 1] = '\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;
        }
}
```