

COMPUTER PROJECT #3

Shaun Harris

Department of Mechanical and Aerospace Engineering
Utah State University
Email: shaun.r.harris@gmail.com

ABSTRACT

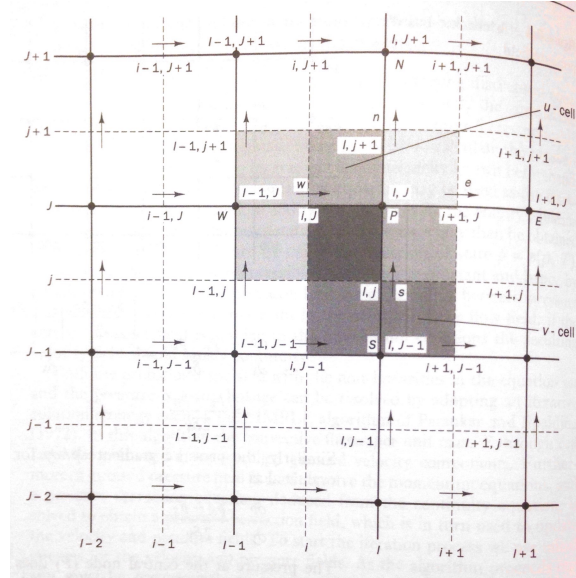
A staggered grid Navier-Stokes solver is implemented to solve a driven cavity problem, and a channel flow problem. The Pressure, u-velocity and v-velocity are all staggered and solved for separately. The necessary equations and the implemented code is provided in this paper.

NOMENCLATURE

| | |
|-------------------|--|
| u | Velocity in the x-direction |
| v | Velocity in the y-direction |
| P | Pressure |
| $a_{i,j}$ | Coefficient for final discretized equation referencing neighbor $i(N, E, S, W, P)$ on $j(u, v, P)$ mesh |
| $\tilde{a}_{P,j}$ | Coefficient for final discretized equation referencing center P on $j(u, v, P)$ mesh and divided by Ω correction factor |
| Ω | non-linear correction factor for momentum equations |
| Ω_P | linear correction factor for pressure equation |
| α | Pressure blending factor |

CONTENTS

| | |
|----------------------------|----------|
| 1 INTRODUCTION | 2 |
| 2 NUMERICAL METHOD | 2 |
| 3 RESULTS | 2 |
| 4 CONCLUSION | 2 |
| A Code | 3 |
| A.1 Subroutines | 3 |
| A.2 Main program | 7 |



Scanned by CamScanner

FIGURE 1. REPRESENTATION OF STENCIL FOR GRID GENERATION

1 INTRODUCTION

In order to solve using this method, a staggered grid was utilized. Fig. 1 shows how the u , v , and P values were saved on the grid. The momentum equation is discretized from Eq. 1 to ??

$$\frac{\partial(\rho uu)}{\partial x} + \frac{\partial(\rho vu)}{\partial y} = -\frac{\partial P}{\partial x} + \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) \quad (1)$$

2 NUMERICAL METHOD

3 RESULTS

4 CONCLUSION

A Code

A.1 Subroutines

```

1  MODULE types
2      !purpose: define data type struct
3      IMPLICIT NONE
4      ! Properties of fluid flow
5      REAL :: Omega = 0.6 ! Relaxation factor for momentum non-linear
6      REAL :: OmegaP= 1.7 ! Relaxation factor for pressure correction linear
7      REAL :: alpha = 0.3 ! relaxation factor for pressure correction
8      REAL :: mu = 0.01 ! dynamic viscosity
9      REAL :: rho= 1. ! density
10     REAL :: Convergence = 1.e-14
11     REAL :: Convergence2= 1.e-9
12     INTEGER :: max_iter = 1000000
13     INTEGER :: max_iter2= 700
14     INTEGER :: max_iter3= 7000
15     TYPE:: dat
16         REAL::xu,yv,xp,yp
17         REAL::u,v,u_old,v_old !u,v is in bottom left corner, or south and west sides of cell
18         REAL::APu,AEu,ANu,ASu,AWu,Apv,AEv,ANv,ASv,AWv,APp,AEp,ANp,AWp,ASp
19         REAL::P,Pp,P_old
20         REAL::S ! source terms
21         INTEGER::n
22     END TYPE dat
23 CONTAINS
24     SUBROUTINE set_xy (strct,dx,dy,nx,ny,x,y)
25         real,intent(in) :: dx,dy,x,y
26         integer,intent(in) :: nx,ny ! size of strct in x and y directions
27         type(dat),dimension(0:,0:),intent(inout)::strct ! data contained from 0:nx-1 where cells 0 and nx-1 are boundary nodes (
28             cell volume approaches 0 on boundary nodes)
29         integer :: i,j,n ! for do loops and n is counter for cell number
30         real :: xi,yi ! x and y values for each cell
31         ! left boundary
32         strct(0,:)%xp= 0.
33         strct(0,0)%yp= 0.
34         strct(0,1:ny-1)%yp= reshape((/ (i*dy - dy/2. ,i=1,ny-1) /),(/ ny-1/))
35         strct(0,ny+0)%yp= y
36         ! bottom boundary
37         strct(0,0)%xp= 0.
38         strct(1:nx-1,0)%xp= reshape((/ (i*dx - dx/2. ,i=1,nx-1) /),(/ nx-1/))
39         strct(nx+0,0)%xp= x
40         strct(:,0)%yp= 0.
41         ! right boundary
42         strct(nx+0,:)%xp= x
43         strct(nx+0,:)%yp= strct(0,:)%yp
44         ! top boundary
45         strct(:,ny+0)%xp= strct(:,0)%xp
46         strct(:,ny+0)%yp= y
47
48         n=1 ! cell number 1
49         DO i=1,ny-1 ! 1 to ny-2 for boundary nodes (we only are iterating through the middle values)
50             yi = i*dy - dy/2. ! y coordinate
51             DO j=1,nx-1
52                 xi = j*dx - dx/2. ! x coordinate
53                 strct(j,i)%n = n ! input n node
54                 strct(j,i)%xp= xi ! x coordinate to strct
55                 strct(j,i)%yp= yi ! y coordinate to strct
56                 n=n+1 ! count cell numbers up one
57             END DO
58         END DO
59         ! set xu and yv to similar values (but for the staggard grids of each)
60         strct%xu = strct%xp - dx/2.
61         strct%yv = strct%yp - dy/2.
62         !top
63         strct(:,ny)%yv=y
64         !bottom
65         strct(:,0)%yv=0.
66         !left
67         strct(0:1,:)%xu=0.
68         !right
69         strct(nx,:)%xu=x
70     END SUBROUTINE set_xy
71
72     SUBROUTINE mom_uv(strct,dx,dy,nx,ny)
73         ! requires uniform grid of dx and dy spacing
74         REAL,INTENT(IN) :: dx,dy

```

```

74 INTEGER,INTENT(IN) :: nx,ny! size of strct in x and y directions
75 TYPE(dat),DIMENSION(0:nx+1,0:ny+1),INTENT(INOUT)::strct ! data contained from 0:nx+1 where cells 0 and nx+1 are boundary
    nodes (cell volume approaches 0 on boundary nodes)
76 REAL :: mdot ! temporary value for mass flow values
77 INTEGER :: i,j,iter=0!loop iterators
78 REAL :: error=1.,error2=1.
79
80 ! mdot and Au values
81 !$OMP PARALLEL DO
82 DO i=1,nx
83     DO j=1,ny
84         mdot = rho*(strct(i+1,j)%u_old+strct(i,j)%u_old)/2.*dy ! east face
85         strct(i,j)%AEu = max(-mdot,0.) + mu*dy/dx
86         mdot = rho*(strct(i-1,j+1)%v_old+strct(i,j+1)%v_old)/2.*dx ! north face
87         IF (j==ny) THEN
88             strct(i,j)%ANu = max(-mdot,0.) + mu*2.*dx/dy
89         ELSE
90             strct(i,j)%ANu = max(-mdot,0.) + mu*dx/dy
91         END IF
92         mdot = rho*(strct(i-1,j)%u_old+strct(i,j)%u_old)/2.*dy ! West face
93         strct(i,j)%AWu = max(mdot,0.) + mu*dy/dx
94         mdot = rho*(strct(i-1,j)%v_old+strct(i,j)%v_old)/2.*dx ! south face
95         IF (j==1) THEN
96             strct(i,j)%ASu = max(mdot,0.) + mu*2.*dx/dy
97         ELSE
98             strct(i,j)%ASu = max(mdot,0.) + mu*dx/dy
99         END IF
100        strct(i,j)%APu = strct(i,j)%AEu + &
101            strct(i,j)%ANu + &
102            strct(i,j)%AWu + &
103            strct(i,j)%ASu
104        strct(i,j)%APu = strct(i,j)%APu/Omega
105    END DO
106 END DO
107 !$OMP END PARALLEL DO
108
109 ! mdot and Av values
110 !$OMP PARALLEL DO
111 DO i=1,nx
112     DO j=1,ny
113         mdot = rho*(strct(i+1,j-1)%u_old+strct(i+1,j)%u_old)/2.*dy ! east face
114         IF (i==nx) THEN
115             strct(i,j)%AEv = max(-mdot,0.) + mu*2.*dy/dx
116         ELSE
117             strct(i,j)%AEv = max(-mdot,0.) + mu*dy/dx
118         END IF
119         mdot = rho*(strct(i,j+1)%v_old+strct(i,j)%v_old)/2.*dx ! north face
120         strct(i,j)%ANv = max(-mdot,0.) + mu*dx/dy
121         mdot = rho*(strct(i,j-1)%u_old+strct(i,j)%u_old)/2.*dy ! West face
122         IF (i==1) THEN
123             strct(i,j)%AWv = max(mdot,0.) + mu*2.*dy/dx
124         ELSE
125             strct(i,j)%AWv = max(mdot,0.) + mu*dy/dx
126         END IF
127         mdot = rho*(strct(i,j-1)%v_old+strct(i,j)%v_old)/2.*dx ! south face
128         strct(i,j)%ASv = max(mdot,0.) + mu*dx/dy
129         strct(i,j)%APv = strct(i,j)%AEv + &
130             strct(i,j)%ANv + &
131             strct(i,j)%AWv + &
132             strct(i,j)%ASv
133         strct(i,j)%APv = strct(i,j)%APv/Omega
134     END DO
135 END DO
136 !$OMP END PARALLEL DO
137
138 ! solve u-momentum
139 error2 = 1.
140 DO iter=1,max_iter
141     error2=error
142     error = 0.
143     DO i=2,nx
144         DO j=1,ny
145             strct(i,j)%u = (1.-Omega)*strct(i,j)%u_old &
146                 + &
147                 (1./strct(i,j)%APu) &
148                 * (&
149                 strct(i,j)%AEu*strct(i+1,j)%u + &

```

```

150         strtct(i ,j )%ANu*strtct(i ,j+1)%u + &
151         strtct(i ,j )%AWu*strtct(i-1,j )%u + &
152         strtct(i ,j )%ASu*strtct(i ,j-1)%u + &
153         (strtct(i-1,j)%P_old-strctct(i ,j)%P_old) &
154         *dy &
155     )
156
157     error = error + (strtct(i,j)%u - strtct(i,j)%u_old)**2
158
159     END DO
160
161     ! strtct(nx+1,:) %u = strtct(nx,:) %u
162     error=sqrt(error)
163     IF (abs(error - error2)<Convergence) EXIT ! error stops changing convergence
164
165     END DO
166
167     WRITE(*,*) sum(rho*dy*strtct(0,:) %u)/sum(rho*dy*strtct(nx+1,:) %u) , iter
168     WRITE(*,*) iter ,abs(error-error2)
169
170     ! solve v-momentum
171     error2 = 1.
172     DO iter=1,max_iter
173         error2=error
174         error = 0.
175         DO i=1,nx
176             DO j=2,ny
177                 strtct(i,j)%v = (1.-Omega)*strtct(i,j)%v_old &
178                 + &
179                 (1./ strtct(i,j)%APv) &
180                 * (&
181                 strtct(i ,j )%AEv*strtct(i+1,j )%v +&
182                 strtct(i ,j )%ANv*strtct(i ,j+1)%v +&
183                 strtct(i ,j )%AWv*strtct(i-1,j )%v +&
184                 strtct(i ,j )%ASv*strtct(i ,j-1)%v +&
185                 (strtct(i,j-1)%P_old-strctct(i ,j)%P_old) *&
186                 dy &
187                 )
188                 error = error + (strtct(i,j)%v - strtct(i,j)%v_old)**2
189             END DO
190         END DO
191         error=sqrt(error)
192         IF (abs(error - error2)<Convergence) EXIT ! error stops changing convergence
193     END DO
194     WRITE(*,*) iter ,abs(error-error2)
195
196     END SUBROUTINE mom_uv
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
SUBROUTINE vel_correction(strct,dx,dy,nx,ny)
! requires uniform grid of dx and dy spacing
REAL,INTENT(IN) :: dx,dy
INTEGER,INTENT(IN) :: nx,ny! size of strtct in x and y directions
TYPE(dat),DIMENSION(0:nx+1,0:ny+1),INTENT(INOUT):: strtct ! data contained from 0:nx+1 where cells 0 and nx+1 are boundary
nodes (cell volume approaches 0 on boundary nodes)
INTEGER :: i,j,iter=0 !loop iterators
REAL :: error,error2
REAL :: S.sum

!$OMP PARALLEL DO
DO i=1,nx
DO j=1,ny
IF (i==nx) THEN
strtct(i,j)%AEp = 0.
ELSE
strtct(i,j)%AEp = rho*dy*dy / strtct(i+1,j)%APu
END IF
IF (j==ny) THEN
strtct(i,j)%ANp = 0.
ELSE
strtct(i,j)%ANp = rho*dx*dx / strtct(i,j+1)%APv
END IF
IF (i==1) THEN
strtct(i,j)%AWp = 0.
ELSE
strtct(i,j)%AWp = rho*dy*dy / strtct(i,j)%APu
END IF
IF (j==1) THEN
strtct(i,j)%ASp = 0.
ELSE
strtct(i,j)%ASp = rho*dx*dx / strtct(i,j)%APv

```

```

226         END IF
227         strect(i,j)%APp = strect(i,j)%AEp + &
228             strect(i,j)%ANp + &
229             strect(i,j)%AWp + &
230             strect(i,j)%ASp
231     END DO
232 END DO
233 !$OMP END PARALLEL DO
234
235 error = 1.
236 error2 = 1.
237 DO iter=1,max_iter2
238     error2=error
239     error=0.
240     S_sum = 0.
241     !$OMP PARALLEL DO
242     DO i=1,nx
243         DO j=1,ny
244             strect(i,j)%S = & ! source terms
245                 (rho*strect(i+1,j)%u-rho*strect(i,j)%u)*dy&
246                 + (rho*strect(i,j+1)%v-rho*strect(i,j)%v)*dx
247         END DO
248     END DO
249     !$OMP END PARALLEL DO
250     DO i=1,nx
251         DO j=1,ny
252             strect(i,j)%Pp = strect(i,j)%Pp + (OmegaP/strect(i,j)%APp)&
253                 *(&
254                 + strect(i,j)%AEp*strect(i+1,j)%Pp&
255                 + strect(i,j)%AWp*strect(i-1,j)%Pp&
256                 + strect(i,j)%ANp*strect(i,j+1)%Pp&
257                 + strect(i,j)%ASp*strect(i,j-1)%Pp&
258                 - strect(i,j)%S &
259                 - strect(i,j)%APp*strect(i,j)%Pp&
260                 )
261         END DO
262     END DO
263     !$OMP PARALLEL DO
264     DO i=1,nx
265         DO j=1,ny
266             strect(i,j)%P=strect(i,j)%P_old+alpha*strect(i,j)%Pp
267         END DO
268     END DO
269     !$OMP END PARALLEL DO
270     DO i=1,nx
271         DO j=1,ny
272             error = error + (strect(i,j)%P - strect(i,j)%P_old)**2
273             S_sum = S_sum + strect(i,j)%S**2
274             IF (ISNAN(strect(i,j)%Pp)) THEN
275                 WRITE(*,*) "error on ",i,j
276                 STOP
277             END IF
278         END DO
279     END DO
280     IF (abs(error - error2)<Convergence) THEN ! error stops changing convergence
281     !IF (abs(S_sum)<Convergence) THEN ! error stops changing convergence
282         !strect%P_old = strect%P
283         EXIT
284     END IF
285 END DO
286 WRITE(*,*) iter,S_sum,abs(error-error2) ! output iterations along with RSS of source term
287 !$OMP PARALLEL DO
288 DO i=2,nx
289     DO j=1,ny
290         strect(i,j)%u=strect(i,j)%u + (strect(i-1,j)%Pp - strect(i,j)%Pp) *dy/strect(i,j)%APu
291     END DO
292 END DO
293 !$OMP END PARALLEL DO
294 !$OMP PARALLEL DO
295 DO i=1,nx
296     DO j=2,ny
297         strect(i,j)%v=strect(i,j)%v + (strect(i,j-1)%Pp - strect(i,j)%Pp) *dx/strect(i,j)%APv
298     END DO
299 END DO
300 !$OMP END PARALLEL DO
301
302 END SUBROUTINE vel_correction

```

```

303 SUBROUTINE Solve_NS(strct,dx,dy,nx,ny)
304   ! requires uniform grid of dx and dy spacing
305   REAL,INTENT(IN)      :: dx,dy
306   INTEGER,INTENT(IN)    :: nx,ny! size of strct in x and y directions
307   TYPE(dat),DIMENSION(0:nx+1,0:ny+1),INTENT(INOUT):: strct ! data contained from 0:nx+1 where cells 0 and nx+1 are boundary
                        nodes (cell volume approaches 0 on boundary nodes)
308   INTEGER :: i,j,iter=0!loop iterators
309   REAL    :: error2=1.,error_RSS=0.
310
311   open(unit=8,file="output/iter.txt")
312   108 FORMAT(2ES16.7)
313   WRITE(8,108) 0.1,1.
314   DO iter=1,max_iter3
315     ! step 1 solve discretised momentum equations
316     CALL mom_uv(strct,dx,dy,nx,ny)
317
318     ! step 2 Solve pressure correction equation
319     ! step 3 Correct pressure and velocities
320     CALL vel_correction(strct,dx,dy,nx,ny)
321
322
323     ! step 4 Solve all other discretised transport equations
324
325     ! if no convergence, then iterate
326     error2 = error_RSS
327     error_RSS = 0.
328     DO i=1,nx
329       DO j=1,ny
330         error_RSS = error_RSS + (strct(i,j)%u-strct(i,j)%u_old)**2
331         error_RSS = error_RSS + (strct(i,j)%v-strct(i,j)%v_old)**2
332         error_RSS = error_RSS + (strct(i,j)%P-strct(i,j)%P_old)**2
333       END DO
334     END DO
335     error_RSS = sqrt(error_RSS)
336     ! reset values
337     strct%u_old = strct%u
338     strct%v_old = strct%v
339     strct%P_old = strct%P
340     !WRITE(*,*) "error = ",error_RSS
341
342
343     ! if converged then stop
344     WRITE(8,108) REAL(iter),abs(error_RSS-error2)
345     IF (abs(error_RSS-error2) <= Convergence2) THEN
346       WRITE(*,*) "converged on iteration and error big loop = ",iter,abs(error_RSS-error2)
347       !WRITE(*,100) ( data(:,i)%S,i=0,max_yp )
348       EXIT
349     ELSE
350       WRITE(*,*) "iteration and error big loop = ",iter,abs(error_RSS-error2)
351     END IF
352   END DO
353   close(8)
354
355
356
357 END SUBROUTINE Solve_NS
358
359
360
361 END MODULE types

```

A.2 Main program

```

1  ! user defined variables to define finite volume
2  ! x and y direction # of cells
3  #define max_x 51
4  #define max_y 51
5  ! x and y number of cells plus 1
6  #define max_xp 52
7  #define max_yp 52
8  ! x and y number of cells plus 2 (to account for boundary nodes)
9  #define max_x2p 53
10 #define max_y2p 53
11 ! length of x and y
12 #define len_x 1.
13 #define len_y 1.
14

```

```

15
16
17 PROGRAM project3
18 USE types !use module defined by types
19 IMPLICIT NONE
20 ! declare variables
21 INTEGER :: i,j!, iter!, max_x=20, max_y=20
22 REAL :: dx, dy
23 REAL :: Lu, Ru, Tu, Bu ! boundary condition u velocity values
24 TYPE(dat), DIMENSION(0:max_xp, 0:max_yp):: data ! 22 if you count edges (thin cell)
25 REAL :: TIME1, TIME2 ! for time of computation
26 ! part 1
27 ! set dx and dy and gamma and coefficients (without dividing by delta x between node centers)
28 dx=len_x/REAL(max_x)
29 dy=len_y/REAL(max_y)
30 ! initialize data and x,y for middle values
31 CALL set_xy(data, dx, dy, max_xp, max_yp, len_x, len_y)
32 !! initialize BC's
33 ! BC's
34 Lu = 0.
35 Ru = 0.
36 Tu = 1.
37 Bu = 0.
38 data%u = 0. ! initialize all data
39 ! left Boundary
40 data(1,:)%u = Lu
41 data(0,:)%u = Lu
42 ! bottom boundary
43 data(:,0)%u = Bu
44 ! right boundary
45 data(max_xp,:)%u = Ru
46 ! top boundary
47 data(:, max_yp)%u = Tu
48 ! initialize u
49 data%u_old = data%u
50
51
52 ! initialize v
53 data%v_old = 0.
54 data%v = 0.
55
56 ! initialize P values
57 data%Pp=0.
58 data%P_old=0.
59 data%P =0.
60
61 ! point SOR method to solve for the exact values of phi using the BC (only loop through inner values)
62 ! solving using the deferred correction method
63 CALL CPU_TIME(TIME1)
64 CALL Solve_NS(data, dx, dy, max_x, max_y)
65 CALL CPU_TIME(TIME2)
66 WRITE(*,*) "CPU Time = ", TIME2-TIME1
67 ! output
68 ! user will need to specify size of
69 open(unit= 9, file="output/x.txt")
70 open(unit=10, file="output/y.txt")
71 open(unit=11, file="output/xu.txt")
72 open(unit=12, file="output/yv.txt")
73 open(unit=13, file="output/u.txt")
74 open(unit=14, file="output/v.txt")
75 open(unit=15, file="output/P.txt")
76 open(unit=16, file="output/u.spot.txt")
77 open(unit=17, file="output/tau_upper.txt")
78 open(unit=18, file="output/tau_lower.txt")
79 open(unit=19, file="output/u.center.txt")
80 100 FORMAT (max_x2p ES16.7)
81 101 FORMAT (2ES16.7)
82 102 FORMAT (max_xp ES16.7)
83 WRITE( 9,100) ( data(:,i)%xp, i=0,max_yp )
84 WRITE(10,100) ( data(:,i)%yp, i=0,max_yp )
85 WRITE(11,100) ( data(:,i)%xu, i=0,max_yp )
86 WRITE(12,100) ( data(:,i)%yv, i=0,max_yp )
87 WRITE(13,100) ( data(:,i)%u , i=0,max_yp )
88 WRITE(14,100) ( data(:,i)%v , i=0,max_yp )
89 WRITE(15,100) ( data(:,i)%P , i=0,max_yp )
90 DO i=0,max_xp
91 ! IF (data(i,1)%xu <= 0.51 .AND. data(i,1)%xu >=0.49) THEN

```



```

92      IF (data(i,1)%xu <= 0.41 .AND. data(i,1)%xu >=0.39) THEN
93          DO j=0,max_yp
94              WRITE(16,101) data(i,j)%u,data(i,j)%yp
95          END DO
96      END IF
97  END DO
98  WRITE(17,102) ( mu*(data(i,max_y)%u-data(i,max_yp)%u)/dy ,i=0,max_x )
99  WRITE(18,102) ( mu*(data(i,1)%u-data(i,0)%u)/dy ,i=0,max_x )
100 DO i=0,max_xp
101     !IF (data(i,1)%xu <= 0.51 .AND. data(i,1)%xu >=0.49) THEN
102     DO j=0,max_yp
103         IF (data(i,j)%yp <= 0.51 .AND. data(i,j)%yp >=0.49) THEN
104             WRITE(19,101) data(i,j)%u,data(i,j)%xp
105         END IF
106     END DO
107 END DO
108 close(9);close(10);close(11);close(12);close(13);close(14);close(15);close(16);close(17);close(18);close(19)
109 END PROGRAM project3

```