Appendix A. Main Program and Initialization

A.1 Main Program

```
PROGRAM connectedcomponents(input,output,infile,outfile,xyfile);
CONST  n=127;      d=3;
       blen=10;      clen=5;
       maxnbr=50;
       nm2=125;     {n-2}
       nm1=126;     {n-1}
       blenm1=9;    {blen-1}
       clenm1=4;    {clen-1}
       upmax1=49;   {maxnbr-1}
       upmax2=99;   {2*maxnbr-1}
       upmax4=199;  {4*maxnbr-1}
       maxnbobj=1000;
```

```
TYPE surroun = 1..2;     {C1}
     adjacen = 1..2;     {C1}
     binary = 0..1;
     max1 = 0..upmax1;
     max2 = 0..upmax2;
     max4 = 0..upmax4;
     m2 = -1..upmax2;
     t03 = 0..3;
     t0maxnbr = 0..maxnbr;
     maxobj = 1..maxnbobj;
     link = ↑objrec;

     objrec=  RECORD
                 num: integer;  {for output}
                 precnnb, succnnb: 0..maxnbr;                        {C2}
                 prefi, prela, sucfi, sucla,                         {C2}
                 preletori, preritole, sucletori, sucritole: link;{C2}
                 fol0, fol1: 1..3;                                   {C2}
                 fol0poin, fol1poin: link;                           {C3}
                 fol0side, fol1side: binary;                         {C3}
                 CASE ty : t03 OF
                 0: ( hro: integer;
                      hbe, hen: 1..nm2 );
                 1: ( fr: integer;
                      b, e: 1..nm2;
                      bl1: 0..blen;
                      blbedif, blendif: ARRAY[0..blenm1] OF -d..d );
                 2: ( ctl: 1..clen;
                      ctbedif, ctendif: ARRAY[0..clenm1] OF -d..d );
                 3: ()
              END;

     runrec=  RECORD
                 objpoin: link;
                 CASE objty:t03 OF
                 0: ( rri: max1 );
                 1: ( rbe, ren: 1..nm2 );
                 2, 3: ()
              END;
     rowrec=  RECORD
                 nbr: 0..maxnbr;
                 runpar: ARRAY[max1] OF runrec
              END;

     cypoin=↑cyrec;

     cyrec=  RECORD
                num: integer;  {for output}
                acces: link;
                whi: binary;
                CASE surroun OF
                1: (pr0, pr1, pl0, pl1: cypoin);
                2: (pr, pl: cypoin);
             END;

VAR surrounding : surroun;
    adjacency : adjacen;
    k : 0..8;
    g, h: binary;
    lp, ls, np, ns, lefpr, lefsu, nripr, nrisu,
    pnp,conpr,consu: max1;
    u: 0..maxnbr;
    v: max4;
    i,ii: integer;
    j: 1..nm2;
    be, en: 1..nm2;
    frow, srow, trow: ARRAY[0..nm1] OF binary;
    blank: runrec;
    thisrow, precrow, emptyrow: rowrec;
    chex: ARRAY[max4] OF m2;
    becs, encs: ARRAY[max4] OF cypoin;     {C4}
    sm : array[max4] of binary;            {C4}
    beho, enho: ARRAY[max2] OF cypoin;     {C5}
    infile,outfile,xyfile : text;
    ncy : integer;

{insert here procedure initialization}
{insert here procedure transitiontothenextrow}
{insert here procedure processonrow}
```

```
BEGIN
open(infile,'data.dat',old);      {C6}
open(outfile,'object.dat',old);   {C6}
open(xyfile,'xy.dat',old);        {C6}
rewrite(outfile);
rewrite(xyfile);
write('enter k (4 or 8 , 0 for exit):');read(k);
WHILE k<>0 DO BEGIN
write('enter surrounding (1 or 2) :');read(surrounding);
write('enter adjacency (1 or 2) :');read(adjacency);
ncy:=0;
reset(infile);
writeln(outfile);writeln(outfile);writeln(outfile);
writeln(outfile,'k=',k:3,'   ***************   surrounding=',
surrounding:3,'   **************   adjacency=',adjacency:3);
writeln(outfile);writeln(outfile);writeln(outfile);
initialization;
i:=1;
WHILE NOT eof(infile) DO
  BEGIN
  processonrow;
  IF NOT eof(infile) THEN transitiontothenextrow;
  i:=i+1
  END;
write('enter k (4 or 8 , 0 for exit):');read(k)
END{while k<>0}
END.
```

A.1.1 Comments

C.1. Variables **surrounding** and **adjacency**, of type **surroun** and **adjacen** respectively, are introduced in order to get a general purpose program:
surrounding = 1 denotes *full surrounding*,
surrounding = 2 denotes *restricted surrounding*,
adjacency = 1 denotes *full adjacency*,
adjacency = 2 denotes *restricted adjacency*.
In the sequel, each time a **CASE** statement relative to adjacency or surrounding appears, only statements corresponding to the case label in question

should appear in a specialized version of the program designed to handle one type surrounding and one type of adjacency.

C.2. Fields to declare under *full adjacency.*

C.3. Fields to declare under *restricted adjacency.*

C.4. Fields to declare under *full surrounding.*

C.5. Fields to declare under *restricted surrounding.*

C.6. The `open` procedure `open(filevar,par1,par2)` opens the file `filevar` according to the VAX/VMS system.

File identifiers `input` and `output` correspond to the display terminal in interractive mode, and allow, in the actual program, to assign variables such that : surrounding, adjacency, k.

File identifier `infile` is an input file containing `X` for a black pixel and blanks otherwise.

File identifier `outfile` is an output file containing the kind of information displayed in Section 6.2.

File identifier `xyfile` is an output file used for graphic applications.

A.2 Initialization

```
PROCEDURE initialization;
VAR x : max1;
BEGIN
IF k=4 THEN h:=0 ELSE h:=1;
g:=1-h;
blank.objpoin:=NIL;
blank.objty:=3;
WITH emptyrow DO
  BEGIN
  nbr:=0;
  FOR x:=0 TO maxnbr-1 DO runpar[x]:=blank;
  END;
thisrow:=emptyrow;
precrow:=emptyrow;
CASE surrounding OF
```

```
     1: FOR v:=0 TO 4*maxnbr-1 DO
        BEGIN
        chex[v]:=-1;
        sm[v]:=0;
        becs[v]:=NIL;
        encs[v]:=NIL
        END;
     2: BEGIN
        FOR v:=0 TO 4*maxnbr-1 DO
        chex[v]:=-1;
        FOR v:=0 TO 2*maxnbr-1 DO
        BEGIN
        beho[v]:=NIL;
        enho[v]:=NIL
        END
        END
END{case surrounding}
END{initialization};
```

Appendix B.
The Processing of Rows

B.1 Procedure Process on Row

```
PROCEDURE processonrow;

{insert here procedure allocate}
{insert here procedure window}

BEGIN
window(i);
u:=0;  lp:=0;  ls:=0;
np:=h*(1-frow[0])*frow[1];
ns:=h*(1-trow[0])*trow[1];
FOR j:=0 TO N-2 DO
  BEGIN
  IF j>0 THEN
    BEGIN
    lp:=lp+(1-frow[j+g])*frow[j-h];
    ls:=ls+(1-trow[j+g])*trow[j-h];
    np:=np+(1-frow[j-g])*frow[j+h];
    ns:=ns+(1-trow[j-g])*trow[j+h]
    END;
  IF (srow[j]=0) AND (srow[j+1]=1)
    THEN
      BEGIN
      be:=j+1;  lefpr:=lp;  lefsu:=ls
      END
    ELSE
      IF (srow[j]=1) AND (srow[j+1]=0) THEN
        BEGIN
        en:=j;  nripr:=np;  nrisu:=ns;
        conpr:=nripr-lefpr;
        consu:=nrisu-lefsu;
        allocate(u);
        u:=u+1;
        pnp:=nripr {pnp=nripr[i,u-1]}
        END
  END
END{processonrow};
```

B.2 Procedure Transition to the Next Row

```
PROCEDURE transitiontothenextrow;
VAR a : binary;
  x,xm,xn : max1;
BEGIN
xm:=thisrow.nbr;
IF xm<precrow.nbr THEN xn:=precrow.nbr ELSE xn:=xm;
precrow.nbr:=xm;
FOR x:=0 TO xn-1 DO
  BEGIN
  precrow.runpar[x]:=thisrow.runpar[x];
  CASE surrounding OF
  1:FOR a:=0 TO 1 DO
    BEGIN
    IF chex[4*x+a]=-1
      THEN chex[4*x+2+a]:=-1
      ELSE chex[4*x+2+a]:=chex[4*x+a]+1;
    sm[4*x+2+a]:=sm[4*x+a];
    becs[4*x+2+a]:=becs[4*x+a];
    encs[4*x+2+a]:=encs[4*x+a]
    END{a};
  2:BEGIN
    FOR a:=0 TO 1 DO
    IF chex[4*x+a]=-1
      THEN chex[4*x+2+a]:=-1
      ELSE chex[4*x+2+a]:=chex[4*x+a]+1;
    beho[2*x+1]:=beho[2*x];
    enho[2*x+1]:=enho[2*x]
    END
  END{case surrounding}
  END{x};
thisrow.nbr:=0;
FOR x:=0 TO xm-1 DO
  BEGIN
  thisrow.runpar[x]:=blank;
  CASE surrounding OF
    1: FOR a:=0 TO 1 DO
       BEGIN
       becs[4*x+a]:=NIL;
       encs[4*x+a]:=NIL
       END;
    2: BEGIN
       beho[2*x]:=NIL;
       enho[2*x]:=NIL;
       END
  END{case surrounding}
  END{x}
END{transitiontothenextrow};
```

B.3 Procedure Window

```
PROCEDURE window(i : integer);
VAR ch : char;
    j : integer;
BEGIN
IF i=1 THEN BEGIN
   FOR j:=0 TO n-1 DO frow[j]:=0;
   FOR j:=0 TO n-1 DO
       BEGIN
       read(infile,ch);
       IF ch = 'X' THEN srow[j]:=1 ELSE srow[j]:=0
       END{j};
   readln(infile);srow[0]:=0;srow[n-1]:=0;
   FOR j:=0 TO n-1 DO
       BEGIN
       read(infile,ch);
       IF ch = 'X' THEN trow[j]:=1 ELSE trow[j]:=0
       END{j};
   readln(infile);trow[0]:=0;trow[n-1]:=0
END{i=1} ELSE
BEGIN
   frow:=srow;
   srow:=trow;
   FOR j:=0 TO n-1 DO
       BEGIN
       read(infile,ch);
       IF ch = 'X' THEN trow[j]:=1 ELSE trow[j]:=0
       END{j};
   readln(infile);trow[0]:=0;trow[n-1]:=0
END{i<>0}
END{window};
```

Appendix C. Procedure Allocate

C.1. Allocate

```
PROCEDURE allocate(u : t0maxnbr);
VAR a : binary;
    p,z : link;
    wrec : runrec;
    q : cypoin;

{insert here procedures operating on cycles i.e.:
          concatenate*  {C1}
          enclose*      {C1}
          extendchain
          newchain
          mergechain
          closechain}

{insert here procedures operating on objects i.e.:
          endof
          conbelow
          newobject
          thisrowobjty0
          thisrowobjty1
          blockenlarge
          continuationenlarge
          newhinge
          newblock
          newcontinuation}

BEGIN
thisrow.nbr:=u+1;
wrec:=precrow.runpar[lefpr];
z:=wrec.objpoin;
IF (conpr=1) AND (consu<=1)
             AND (wrec.objty=1)
             AND (abs(be-wrec.rbe)<=d) {for d-blocks only}
             AND (abs(en-wrec.ren)<=d) {idem}
  THEN
    IF (z↑.ty=1) AND (z↑.bll<blen)
      THEN
        BEGIN
        extendchain(4*lefpr+2,4*u);
        extendchain(4*lefpr+3,4*u+1);
        thisrowobjty1(z);
        blockenlarge(z↑,wrec);
        conbelow(z)
        END
      ELSE
        IF (z↑.ty=2) AND (z↑.ctl<clen)
          THEN
            BEGIN
            extendchain(4*lefpr+2,4*u);
            extendchain(4*lefpr+3,4*u+1);
            thisrowobjty1(z);
            continuationenlarge(z↑,wrec);
            conbelow(z)
            END
          ELSE
            BEGIN
            newobject;
            thisrowobjty1(p);
            newcontinuation(p↑,wrec);
            conbelow(p)
            END
  ELSE
    BEGIN
    newobject;
    IF (conpr<=1) AND (consu<=1)
      THEN
        BEGIN
        thisrowobjty1(p);
        newblock(p↑);
        conbelow(p)
        END
      ELSE
        BEGIN
        thisrowobjty0(p);
        newhinge(p↑);
        conbelow(p)
        END
    END
END{allocate};
```

C.2. Comments

C.1. There exists two versions of each of these procedures: **concatenate1** and **enclose1** handle *full surrounding*, **concatenate2** and **enclose2** handle *restricted surrounding*

Appendix D. The processing of Objects

D.1 Procedure Endof

```
PROCEDURE endof(VAR t:objrec);
VAR bb : 0..blen;
BEGIN
WITH t DO
  BEGIN
  IF (ty=1) AND (bl1<blen)
    THEN
      FOR bb:=bl1 TO blen-1 DO
        BEGIN
        blbedif[bb]:=-(d+1); {or 0}
        blendif[bb]:=-(d+1); {or 0}
        END
    ELSE
      IF (ty=2) AND (ctl<clen)
```

```
        THEN
          FOR bb:=ctl TO clen-1 DO
            BEGIN
            ctbedif[bb]:=-(d+1); {or 0}
            ctendif[bb]:=-(d+1); {or 0}
            END
  END
END{endof};
```

Procedure *endof* is of an "aesthetic" nature. It should be easy to do without it.

D.2 Procedure Conbelow

```
PROCEDURE conbelow(VAR z:link);
BEGIN
CASE adjacency OF
1:BEGIN
z↑.succnnb:=consu;
IF consu=0
  THEN
    BEGIN
    z↑.sucfi:=NIL;      z↑.sucla:=NIL;
    z↑.preletori:=NIL;  z↑.preritole:=NIL;
    z↑.fol0:=1;
    endof(z↑);
    IF chex[4*u]=2*u
      THEN closechain(4*u,4*u+1,z)
      ELSE mergechain(4*u,4*u+1)
    END
END{adjacency =1};
2:BEGIN
IF consu=0
  THEN
    BEGIN
    z↑.fol0poin:=z;  z↑.fol0side:=1;
    endof(z↑);
    IF chex[4*u]=2*u
      THEN closechain(4*u,4*u+1,z)
      ELSE mergechain(4*u,4*u+1)
    END
END{adjacency=2};
END{case adjacency};
END{conbelow};
```

D.3 Procedure Newobject

```
PROCEDURE newobject;
VAR x : max1;
    v : max4;
    s,ss,last : link;
    xrec : runrec;

BEGIN
new(p);     {or anything else with similar effect}
p↑.num:=0;
CASE adjacency OF
1:BEGIN
p↑.precnnb:=conpr;
IF conpr=0
  THEN
    BEGIN
      p↑.prefi:=NIL;      p↑.prela:=NIL;
      p↑.sucletori:=NIL;  p↑.sucritole:=NIL;
      p↑.fol1:=1;
      newchain(4*u,4*u+1)
    END
  ELSE
    FOR x:=0 TO conpr-1 DO
      BEGIN
      xrec:=precrow.runpar[lefpr+x];
      s:=xrec.objpoin;
      endof(s↑);
      IF x=0
        THEN
          BEGIN
          p↑.prefi:=s;
    IF (u>0) AND (lefpr<pnp)
      THEN
        BEGIN
        last:=thisrow.runpar[u-1].objpoin;
        last↑.sucletori:=p;
        p↑.sucritole:=last;
        last↑.foli:=3
        END
      ELSE
        BEGIN
        s↑.sucfi:=p;
        s↑.preritole:=NIL;
        p↑.sucritole:=NIL;
        s↑.fol0:=2;
        extendchain(4*lefpr+2,4*u)
        END
    END {x=0}
  ELSE
    BEGIN
      s↑.sucfi:=p;
      ss↑.preletori:=s;
      s↑.preritole:=ss;
      s↑.fol0:=3;
      v:=4*(lefpr+x)+2;
      IF chex[v]=v DIV 2-2
        THEN closechain(v,v-3,s)
        ELSE mergechain(v,v-3)
    END{x<>0};
IF x=conpr-1
  THEN
    BEGIN
    p↑.prela:=s;
    IF (xrec.objty=1) OR ( (xrec.objty=0) AND (xrec.rri=u) )
      THEN
        BEGIN
          s↑.sucla:=p;
          s↑.preletori:=NIL;
          p↑.sucletori:=NIL;
          p↑.foli:=2;
          extendchain(4*nripr-1,4*u+1)
```

```
              END
            ELSE
              newchain(4*u+4,4*u+1)
          END{x=conpr-1}
        ELSE
          s↑.sucla:=p;
      ss:=s
      END
END{adjacency=1};
2:BEGIN
IF conpr=0
  THEN
    BEGIN
      p↑.folipoin:=p;  p↑.foliside:=0;
      newchain(4*u,4*u+1)
    END
  ELSE
    FOR x:=0 TO conpr-1 DO
      BEGIN
      xrec:=precrow.runpar[lefpr+x];
      s:=xrec.objpoin;
      endof(s↑);
      IF x=0
        THEN
          IF (u>0) AND (lefpr<pnp)
            THEN
              BEGIN
              last:=thisrow.runpar[u-1].objpoin;
              last↑.folipoin:=p; last↑.foliside:=0
              END
            ELSE
              BEGIN
              s↑.fol0poin:=p; s↑.fol0side:=0;
              extendchain(4*lefpr+2,4*u)
              END
        ELSE
          BEGIN
          s↑.fol0poin:=ss;  s↑.fol0side:=1;
          v:=4*(lefpr+x)+2;
          IF chex[v]=v DIV 2-2
```

```
            THEN closechain(v,v-3,s)
            ELSE mergechain(v,v-3)
          END;
      IF x=conpr-1
        THEN
         BEGIN
         IF (xrec.objty=1) OR ( (xrec.objty=0) AND (xrec.rri=u) )
           THEN
             BEGIN
             p↑.folipoin:=s; p↑.foliside:=1;
             extendchain(4*nripr-1,4*u+1)
             END
           ELSE
             newchain(4*u+4,4*u+1)
         END;
    ss:=s
    END;
END{adjacency=2}
END{case adjacency}
END{newobject};
```

At this point, it is useful to recall from Section 4.5 that $objty = 0$ for a hinge, while $objty = 1$ for block-runs belonging to both types 1 and 2 objects.

D.4 Procedure Thisrowobjty0

```
PROCEDURE thisrowobjty0(VAR z: link);
BEGIN
WITH thisrow.runpar[u] DO
  BEGIN
  objpoin:=z;
  objty:=0;          {for a hinge}
  rri:=nrisu-1
  END
END{thisrowobjty0};
```

D.5 The Procedure Thisrowobjty1

```
PROCEDURE thisrowobjty1(VAR z: link);
BEGIN
WITH thisrow.runpar[u] DO
  BEGIN
  objpoin:=z;
  objty:=1;          {for a block run}
  rbe:=be;
  ren:=en
  END
END{thisrowobjty1};
```

D.6 Procedure Blockenlarge

```
PROCEDURE blockenlarge(VAR t: objrec; VAR w: runrec);
BEGIN
WITH t DO
  BEGIN
  blbedif[bl1]:=be-w.rbe;
  blendif[bl1]:=en-w.ren;
  bl1:=bl1+1
  END
END{blockenlarge};
```

D.7 Procedure Continuationenlarge

```
PROCEDURE continuationenlarge(VAR t: objrec; VAR w: runrec);
BEGIN
WITH t DO
  BEGIN
  ctbedif[ctl]:=be-w.rbe;
  ctendif[ctl]:=en-w.ren;
  ctl:=ctl+1
  END
END{continuationenlarge};
```

D.8 Procedure Newhinge

```
PROCEDURE newhinge(VAR t: objrec);
BEGIN
WITH t DO
  BEGIN
  ty:=0;
  hro:=i;  hbe:=be;  hen:=en
  END
END{newhinge};
```

D.9 Procedure Newblock

```
PROCEDURE newblock(VAR t: objrec);
BEGIN
WITH t DO
  BEGIN
  ty:=1;
  fr:=i;  b:=be;  e:=en;
  bll:=0
  END
END{newblock};
```

D.10 Procedure Newcontinuation

```
PROCEDURE newcontinuation(VAR t: objrec; VAR w: runrec);
BEGIN
WITH t DO
  BEGIN
  ty:=2;
  ctbedif[0]:=be-w.rbe;
  ctendif[0]:=en-w.ren;
  ctl:=1
  END
END{newcontinuation};
```

Appendix E. The Processing of Cycles

In this appendix, the notation wi corresponds to w_i in Subsection 5.4.4.

E.1 Procedure Concatenate1

```
PROCEDURE concatenate1(VAR w0:max4; w1: max4);

{Procedure concatenate1 operates under full surrounding.}
{We must have (w0-w1) MOD 2=0.}

BEGIN
IF becs[w0]<>NIL
  THEN
    BEGIN
    IF becs[w1]=NIL
      THEN
        encs[w1]:=encs[w0]
```

```
   ELSE
    IF becs[w0]↑.whi=1
     THEN
      BEGIN
      encs[w0]↑.pr0:=becs[w1];
      becs[w1]↑.pl1:=encs[w0]
      END
     ELSE
      BEGIN
      encs[w0]↑.pr1:=becs[w1];
      becs[w1]↑.pl0:=encs[w0]
      END;
  becs[w1]:=becs[w0];
  encs[w0]:=NIL; becs[w0]:=NIL
  END
END{concatenate1};
```

E.2 Procedure Concatenate2

```
PROCEDURE CONCATENATE2(y0, y1: max2);

{Procedure concatenate2 operates under restricted surrounding.}

BEGIN
IF beho[y0]<>NIL
 THEN
  BEGIN
  IF beho[y1]=NIL
   THEN
    enho[y1]:=enho[y0]
   ELSE
    BEGIN
    enho[y0]↑.pr:=beho[y1];  beho[y1]↑.pl:=enho[y0]
    END;
  beho[y1]:=beho[y0];
  enho[y0]:=NIL;  beho[y0]:=NIL
  END
END{concatenate2};
```

E.3 Procedure Enclose1

```
PROCEDURE enclose1(VAR w: max4; VAR q: cypoin; a: binary);

{Procedure enclose1 operates in full surrounding.}

BEGIN
q↑.whi:=a;
IF becs[w]=NIL
  THEN
    IF a=0
      THEN
        BEGIN
        q↑.pr0:=q; q↑.pl1:=q
        END
      ELSE
        BEGIN
        q↑.pr1:=q; q↑.pl0:=q
        END
  ELSE
    BEGIN
    IF a=0
      THEN
        BEGIN
        q↑.pr0:=becs[w];
        becs[w]↑.pl1:=q;
        q↑.pl1:=encs[w];
        encs[w]↑.pr0:=q
        END
      ELSE
        BEGIN
        q↑.pr1:=becs[w];
        becs[w]↑.pl0:=q;
        q↑.pl0:=encs[w];
        encs[w]↑.pr1:=q
        END;
    becs[w]:=NIL;  encs[w]:=NIL
    END
END{enclose1};
```

E.4 Procedure Enclose2

```
PROCEDURE enclose2(y: max2; VAR q: cypoin);

{Procedure enclose2 operates in restricted surrounding.}

BEGIN
IF beho[y]=NIL
  THEN
    BEGIN
    q↑.pr:=q;  q↑.pl:=q
    END
  ELSE
    BEGIN
    q↑.pr:=beho[y];  beho[y]↑.pl:=q;
    q↑.pl:=enho[y];  enho[y]↑.pr:=q;
    beho[y]:=NIL;    enho[y]:=NIL
    END
END{enclose2};
```

E.5 Procedure Extenchain

```
PROCEDURE extendchain(w0, w1: max4);
VAR w2: max4;
BEGIN
IF odd(w0) THEN w2:= 2*chex[w0]
           ELSE w2:= 2*chex[w0]+1;
chex[w1]:=chex[w0];
chex[w0]:=-1;
chex[w2]:=w1 DIV 2;
CASE surrounding OF
1:BEGIN
sm[w1]:=sm[w0];
IF odd(w0)
  THEN
    BEGIN
    becs[w1]:=becs[w0];  encs[w1]:=encs[w0];
    becs[w0]:=NIL;          encs[w0]:=NIL
    END
  ELSE
    concatenate1(w0,w1);
  END{surrounding=1};
2:BEGIN
IF odd(w0)
  THEN
    BEGIN
    beho[w1 DIV 2]:=beho[w0 DIV 2];
    enho[w1 DIV 2]:=enho[w0 DIV 2];
    beho[w0 DIV 2]:=NIL;  enho[w0 DIV 2]:=NIL
    END
END{surrounding=2}
END{case surrounding}
END{extendchain};
```

E.6 Procedure Newchain

```
PROCEDURE newchain(w0, w1: max4);
BEGIN
chex[w0]:=w1 DIV 2;
chex[w1]:=w0 DIV 2;
IF surrounding=1 THEN
BEGIN
IF w0>w1
  THEN
    sm[w0]:=0
  ELSE
    IF w0=0
      THEN
        sm[w0]:=1
      ELSE
        sm[w0]:=sm[w0-3];
sm[w1]:=sm[w0]
END{surrounding=1}
END{newchain};
```

E.7 Procedure Mergechain

```
PROCEDURE mergechain(w0, w1: max4);
VAR w2, w3: max4;
BEGIN
w2:=2*chex[w1];  w3:=2*chex[w0]+1;
chex[w2]:=chex[w0];  chex[w3]:=chex[w1];
chex[w0]:=-1;  chex[w1]:=-1;
CASE surrounding OF
1:BEGIN
IF sm[w0]<sm[w1]
  THEN
    BEGIN
    sm[w0]:=sm[w1];  sm[w3]:=sm[w1]
    END
  ELSE
    BEGIN
    sm[w1]:=sm[w0];  sm[w2]:=sm[w0]
    END;
concatenate1(w1,w3);
concatenate1(w0,w2)
END{surrounding=1};
2:concatenate2(w1 DIV 2,w3 DIV 2);
END{case surrounding}
END{mergechain};
```

E.8 Procedure Closechain

```
PROCEDURE closechain(w0, w1: max4; VAR z: link);

{insert here procedure outcy2}
{insert here procedure outcy1}

BEGIN
new(q);  {or anything else with similar effect}
q↑.acces:=z;
q↑.num:=0;
```

```
IF w0<w1
  THEN
    BEGIN
    q↑.whi:=0;
CASE surrounding OF
1:BEGIN
    enclose1(w1,q,0);

    {The connected component enclosed by q↑ is completely disclosed.}

    IF sm[w0]=0
      THEN                                {the left edge w0+4 exists}
        BEGIN
        becs[w0+4]:=q;  encs[w0+4]:=q;
        concatenate1(w0,w0+4);
        outcy1(q,false)
        END
      ELSE outcy1(q,true);

        {if sm[w0]=1, then becs[w0]=NIL and the string enclosed by q
        is maximal}

END{surrounding=1};
2:BEGIN
  enclose2(w1 DIV 2,q);

  {The connected component enclosed by q↑ is completely disclosed.}

  outcy2(q)
  END
END{case surrounding};
    END{w0<w1}
  ELSE
    BEGIN
    q↑.whi:=1;
CASE surrounding OF
1:BEGIN
    enclose1(w0,q,1);
    concatenate1(w1,w0+1);
    becs[w1]:=q;  encs[w1]:=q;
    concatenate1(w1,w0+1);
    ncy:=ncy+1;   q↑.num:=ncy
END{surrounding=1};
2:BEGIN
    concatenate2(w1 DIV 2,w0 DIV 2);
    beho[w1 DIV 2]:=q;  enho[w1 DIV 2]:=q;
    concatenate2(w1 DIV 2,w0 DIV 2)
END{surrounding=2}
END{case surrounding}
    END
END{closechain};
```

Appendix F.
Output Procedures

F.1 Procedures Outcy

F.1.1 Procedure Outcy1

```
PROCEDURE outcy1(VAR q : cypoin; max : boolean);

{Procedure outcy1 operates in full surrounding.}
VAR c1,c2,c : cypoin;
    vp : ARRAY[maxobj] OF link;
    vpcy : ARRAY[maxobj] OF integer;
    p,p1 : link;
    ivp,nvp : integer;
    side : binary;
    instring : boolean;

{insert here procedure outobj1}
```

```
{insert here procedure interncy}
{insert here procedure idobj1}
{insert here procedure outxy}

BEGIN
nvp:=0; side:=0; c1:=q;
ncy:=ncy+1; q↑.num:=ncy;
REPEAT
p:=c1↑.acces;p1:=p;
CASE adjacency OF
   1:REPEAT
       idobj1(p);
       IF side=0 THEN
              CASE p↑.fol0 OF
                 1:side:=1;
                 2:p:=p↑.sucfi;
                 3:BEGIN side:=1; p:=p↑.preritole END;
              END{case p↑.fol0}
                  ELSE
              CASE p↑.fol1 OF
                  1:side:=0;
                  2:p:=p↑.prela;
                  3:BEGIN side:=0; p:=p↑.sucletori END;
              END{case p↑.fol1};
      UNTIL ((p=p1) AND (side=0));
     2:REPEAT
       idobj1(p);
       IF side=0 THEN BEGIN
                      side:=p↑.fol0side;
                      p:=p↑.fol0poin
                      END
                 ELSE BEGIN
                      side:=p↑.fol1side;
                      p:=p↑.fol1poin
                      END;
      UNTIL ((p=p1) AND (side=0))
  END{case adjacency};
  c1:=c1↑.pr0;
UNTIL c1=q;
outobj1(vp,nvp);
```

```
outxy(vp,nvp);
FOR ivp:=1 TO nvp DO dispose(vp[ivp]);
                                {or anything else with similar effect}
IF max THEN
BEGIN                           {cycle c1 is maximal}
writeln(outfile,'MAXIMAL COMPONENT : ',c1↑.num:3);
interncy(c1);
c:=c1;instring:=true;
WHILE instring DO BEGIN
   IF c↑.num=-1 THEN
      BEGIN
      IF c=c1 THEN instring:=false ELSE
      IF c↑.whi=0 THEN c2:=c↑.pr1 ELSE c2:=c↑.pr0;
      dispose(c)
      END
   ELSE
      BEGIN
      IF c↑.whi=0 THEN c2:=c↑.pr0 ELSE c2:=c↑.pr1;
      writeln(outfile,'dispose ',c↑.num:3);
      c↑.num:=-1
      END;
   c:=c2
   END;
END{max}
END{outcy1};
```

F.1.2 Procedure Outcy2

```
PROCEDURE outcy2(VAR q : cypoin);

{Procedure outcy2 operates under restricted surrounding.}

VAR c1,c2 : cypoin;
    vp : ARRAY[maxobj] OF link;
    p,p1 : link;
    ivp,nvp,nhole : integer;
    side : binary;
```

```
{insert here procedure outobj2}
{insert here procedure idobj2}
{insert here procedure outxy}

BEGIN
nvp:=0; side:=0; c1:=q;
REPEAT
p:=c1↑.acces;p1:=p;
CASE adjacency OF
   1:REPEAT
       idobj2(p);
       IF side=0 THEN
              CASE p↑.fol0 OF
                 1:side:=1;
                 2:p:=p↑.sucfi;
                 3:BEGIN side:=1; p:=p↑.preritole END
              END{case p↑.fol0}
                  ELSE
              CASE p↑.fol1 OF
                  1:side:=0;
                  2:p:=p↑.prela;
                  3:BEGIN side:=0; p:=p↑.sucletori END
              END{case p↑.fol1};
      UNTIL ((p=p1) AND (side=0));
     2:REPEAT
       idobj2(p);
       IF side=0 THEN BEGIN
                      side:=p↑.fol0side;
                      p:=p↑.fol0poin
                      END
                 ELSE BEGIN
                      side:=p↑.fol1side;
                      p:=p↑.fol1poin
                      END;
      UNTIL ((p=p1) AND (side=0))
  END{case adjacency};
  c1:=c1↑.pr;
UNTIL c1=q;
outobj2(vp,nvp);
outxy(vp,nvp);
```

```
FOR ivp:=1 TO nvp DO dispose(vp[ivp]);
c1:=q↑.pr;nhole:=0;
WHILE c1<>q DO BEGIN
nhole:=nhole+1;
c2:=c1↑.pr;dispose(c1);c1:=c2 END;
dispose(q);
writeln(outfile);
writeln(outfile,'end of a component containing ',nhole:2,' holes');
writeln(outfile);
END {outcy2};
```

F.2 Procedures Outobj

F.2.1 procedure Outobj1

```
PROCEDURE outobj1(VAR vp : ARRAY[maxobj] OF link; VAR nvp : integer);

{Procedure outobj1 operates under full surrounding.}

VAR i,j : integer;
FUNCTION valnum(VAR p:link) : integer;
BEGIN
IF p<>NIL THEN valnum:=p↑.num
ElSE valnum:=0
END{valnum};
BEGIN
FOR i:=1 TO nvp DO
WITH vp[i]↑ DO
BEGIN
writeln(outfile,'object:',i:3,'---------', 'cycle :',vpcy[i]:3);
CASE adjacency OF
1:BEGIN
   writeln(outfile,'precnnb=',precnnb:3,
               '     succnnb=',succnnb:3);
   writeln(outfile,'prefi=',valnum(prefi):3,
               '     prela=',valnum(prela):3,
               '    sucfi=',valnum(sucfi):3,
               '    sucla=',valnum(sucla):3);
   writeln(outfile,'preletori=',valnum(preletori):3,
               '    preritole=',valnum(preritole):3,
               '    sucletori=',valnum(sucletori):3,
               '    sucritole=',valnum(sucritole):3);
   writeln(outfile,'fol0=',fol0:3,
               '    fol1=',fol1:3)
  END{1};
2:BEGIN
   writeln(outfile,'fol0poin=',fol0point↑.num:3,
               '    fol1poin=',fol1point↑.num:3);
   writeln(outfile,'fol0side=',fol0side:3,
               '    fol1side=',fol1side:3)
  END{2}
END{case adjacency};
writeln(outfile,'ty=',ty:1);
CASE ty OF
0:writeln(outfile,'hro=',hro:3,
              '    hbe=',hbe:3,
              '    hen=',hen:3);
1:BEGIN
   writeln(outfile,'fr=',fr:3,
                '    b=',b:3,
                '    e=',e:3,
                '  bll=',bll:3);
   write(outfile,' ');
   FOR j:=0 TO bll-1 DO write(outfile,'(',j:1,')');
   FOR j:=0 TO bll-1 DO write(outfile,blbedif[j]:3);
   FOR j:=0 TO bll-1 DO write(outfile,blendif[j]:3)
  END{1};
2:BEGIN
   writeln(outfile,'ctl=',ctl:3);write(outfile,' ');
   FOR j:=0 TO ctl-1 DO write(outfile,'(',j:1,')');
   FOR j:=0 TO ctl-1 DO write(outfile,ctbedif[j]:3);
   FOR j:=0 TO ctl-1 DO write(outfile,ctendif[j]:3)
  END{2}
END {case ty}
END{with vp[i]↑}
END{outobj1};
```

F.2.2 Procedure Outobj2

```
PROCEDURE outobj2(VAR vp : ARRAY[maxobj] OF link;
                  VAR nvp : integer);

{Procedure outobj2 operates under restricted surrounding.}

VAR i,j : integer;
FUNCTION valnum(VAR p:link) : integer;
BEGIN
IF p<>NIL THEN valnum:=p↑.num
ELSE valnum:=0
END{valnum};
BEGIN
FOR i:=1 TO nvp DO
WITH vp[i]↑ DO
BEGIN
writeln(outfile,'object:',i:3,'----------');
CASE adjacency OF
1:BEGIN
   writeln(outfile,'precnnb=',precnnb:3,
                 '     succnnb=',succnnb:3);
   writeln(outfile,'prefi=',valnum(prefi):3,
                 '     prela=',valnum(prela):3,
                 '     sucfi=',valnum(sucfi):3,
                 '     sucla=',valnum(sucla):3);
   writeln(outfile,'preletori=',valnum(preletori):3,
                 '     preritole=',valnum(preritole):3,
                 '     sucletori=',valnum(sucletori):3,
                 '     sucritole=',valnum(sucritole):3);
   writeln(outfile,'fol0=',fol0:3,
                 '     fol1=',fol1:3)
  END{1};
2:BEGIN
   writeln(outfile,'fol0poin=',fol0point↑.num:3,
                 '     fol1poin=',fol1point↑.num:3);
   writeln(outfile,'fol0side=',fol0side:3,
                 '     fol1side=',fol1side:3)
  END{2}
```

```
END{case adjacency};
writeln(outfile,'ty=',ty:1);
CASE ty OF
0:writeln(outfile,'hro=',hro:3,
             '     hbe=',hbe:3,
             '     hen=',hen:3);
1:BEGIN
   writeln(outfile,'fr=',fr:3,'     b=',b:3,
                '     e=',e:3,'    bll=',bll:3);
   write(outfile,' ');
   FOR j:=0 TO bll-1 DO write(outfile,'(',j:1,')');
   FOR j:=0 TO bll-1 DO write(outfile,blbedif[j]:3);
   FOR j:=0 TO bll-1 DO write(outfile,blendif[j]:3)
  END{1};
2:BEGIN
   writeln(outfile,'ctl=',ctl:3);write(outfile,' ');
   FOR j:=0 TO ctl-1 DO write(outfile,'(',j:1,')');
   FOR j:=0 TO ctl-1 DO write(outfile,ctbedif[j]:3);
   FOR j:=0 TO ctl-1 DO write(outfile,ctendif[j]:3)
  END{2}
END {case ty}
END{with vp[i]↑}
END{outobj2};
```

F.3 Procedures Idobj

F.3.1 Procedure Idobj1

```
PROCEDURE idobj1(VAR p : link);

{Procedure idobj1 operates under full surrounding.}

BEGIN
IF p↑.num=0 THEN
BEGIN
  nvp:=nvp+1;  vp[nvp]:=p;
  vpcy[nvp]:=ncy;  p↑.num:=nvp
END
END{idobj1};
```

F.3.2 Procedure Idobj2

```
PROCEDURE idobj2(VAR p : link);

{Procedure idobj2 operates under restricted surrounding.}

BEGIN
IF p↑.num=0 THEN
BEGIN
  nvp:=nvp+1;  vp[nvp]:=p;  p↑.num:=nvp
END
END{idobj2};
```

F.4 Procedure Outxy

```
PROCEDURE outxy(VAR vp : ARRAY[maxobj] OF link; VAR nvp : integer);
VAR k,ii : integer;
    j,c1,c2 : 1..nm2;
    iblen : 0..blen;
    iclen : 1..clen;
    cont : link;
BEGIN
FOR k:=1 TO nvp DO
WITH vp[k]↑ DO
IF ty=0 THEN
   BEGIN
   FOR j:=hbe TO hen DO
   writeln(xyfile,hro,j)
   END
ELSE
IF ty=1 THEN
   BEGIN
   c1:=b;c2:=e;
   FOR j:=b TO e DO writeln(xyfile,fr,j);
   FOR iblen:=0 TO bll-1 DO
      BEGIN
      c1:=c1+blbedif[iblen];
      c2:=c2+blendif[iblen];
      FOR j:=c1 TO c2 DO writeln(xyfile,fr+iblen+1,j)
      END;
   CASE adjacency OF
   2: BEGIN
      cont:=vp[k];ii:=fr+bll;
      WHILE (cont↑.fol0side=0) AND (cont↑.fol0point↑.ty=2)  DO
         BEGIN
         cont:=cont↑.fol0poin;
         FOR iclen:=1 TO cont↑.ctl DO
            BEGIN
            c1:=c1+cont↑.ctbedif[iclen-1];
            c2:=c2+cont↑.ctendif[iclen-1];
            FOR j:=c1 TO c2 DO writeln(xyfile,ii+iclen,j)
            END;
         ii:=ii+cont↑.ctl
         END
      END {adjacency=2};
   1: BEGIN
      cont:=vp[k];ii:=fr+bll;
      WHILE (cont↑.fol0=2) AND (cont↑.sucfi↑.ty=2) DO
         BEGIN
         cont:=cont↑.sucfi;
         FOR iclen:=1 TO cont↑.ctl DO
            BEGIN
            c1:=c1+cont↑.ctbedif[iclen-1];
            c2:=c2+cont↑.ctendif[iclen-1];
            FOR j:=c1 TO c2 DO writeln(xyfile,ii+iclen,j)
            END;
         ii:=ii+cont↑.ctl
         END
      END {adjacency=1}
   END{case adjacency}
END{ty=1}
END{outxy};
```

F.5 Procedure Interncy

```
PROCEDURE interncy(c : cypoin);
VAR c1 : cypoin;
BEGIN
c1:=c;
IF c↑.whi = 0 THEN
   BEGIN
   WHILE c↑.pr0<>c1 DO
      BEGIN
      c:=c↑.pr0;
      writeln(outfile,c1↑.num:3,'    ---> ',c↑.num:3,' hole');
      interncy(c)
      END
   END
ELSE
   BEGIN
      WHILE c↑.pr1<>c1 DO
      BEGIN
      c:=c↑.pr1;
      writeln(outfile,c1↑.num:3,'    ---> ',c↑.num:3);
      interncy(c)
      END
   END
END{interncy};
```