Se incluye a continuación un pseudocódigo por cada uno de los procedimientos que se deben implementar en esta práctica. Para aquellos datos usados que no están definidos como operandos se deja libertad de utilizar registros o variables locales, aunque en algunos casos sea obligatorio el uso de registros (cuando los datos se usen para direccionar la memoria). En cualquier caso, se recuerda que en la arquitectura de 64 bits hay 8 nuevos registros de propósito general que pueden utilizarse como almacenes de datos auxiliares, evitando el uso de variables locales.

```
void borrar(uchar * imgD, int w, int h)
{
    dirDest = imgD;
    tam = w*h
    Para(p=0; p<tam; p++)
    {
        [dirDest] = 0;
        dirDest++;
    }
}

void invertir(uchar * imgO, uchar * imgD, int w, int h)
{
    dirOrig = imgO;
    dirDest = imgD;
    tam = w*h
    Para(p=0; p<tam; p++)
    {
        [dirDest] = 255-[dirOrig];
        dirOrig++;
        dirDest++;
    }
}

void umbralizar(uchar * imgD, int w, int h)
{
    dirDest = imgD;
    tam = w*h
    Para(p=0; p<tam; p++)
    {
        Si([dirDest]>128)
            [dirDest] = 255;
        Sino
            [dirDest] = 0;
        dirDest++;
    }
}
```

```
void eliminarRuido_F1(uchar * imgO, uchar * imgD, int w, int h)
{
    dirOrig = imgO;
    dirDest = imgD;
    Para(f=1; f<h-1; f++)
    {
        Para(c=1; c<w-1; c++)
        {
            df=-1;
            tBlancos = true;
            Mientras(df<=1 y tBlancos)
            {
                dc=-1;
                Mientras(dc<=1 y tBlancos)
                {
                    posPixel = (f+df)*w + (c+dc);
                    Si([dirOrig+posPixel]==0)
                        tBlancos=false;
                    dc++
                }
                df++;
            }
            posPixel = f*w + c;
            Si(tBlancos)
                [dirDest+posPixel] = 255;
            Sino
                [dirDest+posPixel] = 0;
        }
    }
}

void eliminarRuido_F2(uchar * imgO, uchar * imgD, int w, int h)
{
    dirOrig = imgO;
    dirDest = imgD;
    Para(f=1; f<h-1; f++)
    {
        Para(c=1; c<w-1; c++)
        {
            df=-1;
            aBlanco = false;
            Mientras(df<=1 y no aBlanco)
            {
                dc=-1;
                Mientras(dc<=1 y no aBlanco)
                {
```

```
                    posPixel = (f+df)*w + (c+dc);
                    Si([dirOrig+posPixel]==255)
                        aBlanco=true;
                    dc++
                }
                df++;
            }
            posPixel = f*w + c;
            Si(aBlanco)
                [dirDest+posPixel] = 255;
            Sino
                [dirDest+posPixel] = 0;
        }
    }

}

int detectarV_min(uchar *imgD, int U)
{
    //Modifica la variable local "min", cuyo valor es retornado
    //por el procedimiento

    dirDest = imgD;
    f = 0;
    cont = 0;
    Mientras(f<100 y cont<=U)
    {
        cont = 0;
        Para(c=0; c<320; c++)
        {
            Si([dirDest]==255)
                cont++;
            dirDest++;
        }
        f++
    }
    min = f-1;

}
```

```
int detectarV_max(uchar *imgD, int U)
{
    //Modifica la variable local "max", cuyo valor es retornado
    //por el procedimiento

    dirDest = imgD + 320*100-1;
    f = 99;
    cont = 0;
    Mientras(f>=0 y cont<=U)
    {
        cont = 0;
        Para(c=0; c<320; c++)
        {
            Si([dirDest]==255)
                cont++;
            dirDest--;
        }
        f--
    }
    max = f+1;

}

void detectarH_F1(uchar *imgD, uchar * VA, uchar U)
{
    dirVA = VA;

    Para(c=0; c<320; c++)
    {
        dirDest = imgD;
        cont = 0;
        Para(f=0; f<100; f++)
        {
            Si([dirDest+c]==255)
                cont++;
            dirDest = dirDest+320;
        }
        Si(cont>U)
            [dirVA+c] = 1;
        Sino
            [dirVA+c] = 0;

    }

}
```

```
void detectarH_F2(uchar * VA, int * Vh)
{
    dirVA = VA;
    dirVh = Vh;

    c = 0;
    nC = 0;
    Mientras(c<320 y nC<7)
    {
        Si([dirVA+c] == 1)
        {
            iC = c;
            iC++;
            Mientras(iC<320 y [dirVA+iC] == 1)
                iC++;
            tamC = iC-c;
            [dirVh] = c + tamC/2;
            dirVh = dirVh+4;
            nC++;
            c = iC-1;
        }
        c++
    }

}
```

```
void recortar(uchar *imgO, uchar *imgD,int x, int y, int w, int h)
{
    dirOrig = imgO + 320*y+x;
    dirDest = imgD;

    Para(f=0; f<h; f++)
    {
        Para(c=0; c<w; c++)
        {
            [dirDest] = [dirOrig]
            dirDest++;
            dirOrig++
        }
        dirOrig = dirOrig + 320 - w;
    }

}
```

```
int matching(uchar *caracM, uchar *patrones[31], int ini, int tam)
{
    //Modifica la variable local "iMax", cuyo valor es retornado
    //por el procedimiento

    dirPatrones = patrones
    iP = ini;
    maxSim = -1000
    Para(nP=0; nP<tam; nP++)
    {
        posPatron = iP*8;
        dirP = [dirPatrones + posPatron];
        dirCarac = caracM;
        sim = 0;
        Para(p=0; p < 32*55; p++)
        {
            Si([dirCarac] == [dirP])
            {
                Si([dirCarac] = 255)
                    sim = sim+3;
                Sino
                    sim = sim+1;

            }
            Sino
                sim = sim-1;

        }
        Si(sim>maxSim)
        {
            maxSim = sim;
            iMax = iP;
        }
        iP++;
    }
}
```