Reconhecimento de algarismos manuais

Trabalho realizado por:

Filipe Moreira

Identificação de algarismos manuais isolados

1. Objectivos

Este trabalho teve como objectivo desenvolver algoritmos e aplicá-los a um programa capaz de reconhecer algarismos isolados escritos manualmente.

2. Descrição do problema

Os algarismos, quando escritos manualmente, não são sempre feitos da mesma maneira, o que impossibilita determinar a localização dos pixels que tenham um dado valor diferente do valor dos pixels que constituem o fundo, para, automaticamente, reconhecer esses algarismos; tal não acontece com os caracteres escritos num computador, por exemplo, em que, para cada tipo de fonte, é bem conhecida a localização de cada pixel pertencente a esse caracter.
Assim sendo, é necessário elaborar algoritmos que tenham em atenção as características de cada algarismos e tentar, de algum modo, reconhecer se essas características estão presentes. Identificar essas características, torna-se, assim, o principal ponto do problema.

3. Desenvolvimento de algoritmos

3.1 Obtenção das imagens

O formato de ficheiros utilizado para a captação das imagens foi o BITMAP (*.bmp), de acordo com o *software* fornecido pelo Docente da disciplina. Esses ficheiros teriam de ter 8 bits por pixel e ser monocromáticos (escala de cinzentos, em 256 níveis - intensidade).
Para alcançar este objectivo foi necessário estudar o tipo de ficheiros BITMAP. Este tipo de ficheiros é constituído por um primeiro cabeçalho – BITMAPFILEHEADER – que contem dados relativos ao ficheiro, desde a identificação do tipo ('BM'), bem como o tamanho e outros dados; segue-se a este cabeçalho um outro cabeçalho com informações relativamente à imagem de bits, tais como a sua altura, largura, o número de bits utilizados por pixel, o número de pixels por comprimento (quer para a altura, quer para a largura), o tipo de compressão usada, o número de cores, etc. Estes cabeçalhos têm uma dimensão de 14 mais 40 bytes no total. Segue-se, em alguns casos – sempre que se usem 8 ou menos bits por pixel – uma tabela RGBQUAD. Por fim aparecem os valores, propriamente ditos, dos níveis de intensidade de cada pixel. De notar que estes valores estão escritos linha a linha de baixo para cima e têm de perfazer uma fronteira de 32 bytes.

Assim sendo, depois de descodificado este tipo de ficheiros, retiraram-se os valores de intensidade para cada pixel e colocaram-se esse valores numa matriz, que representa, deste modo, o mapeamento da imagem.

3.2 Adelgaçamento dos objectos

Os algarismos, normalmente, têm mais do que um pixel de espessura, pelo que tentar determinar as características em tais circunstâncias seria imprudente; assim sendo, recorre-se, inicialmente, a uma operação de adelgaçamento de objectos negros assume-se que os algarismos serão escritos em folhas de papel claro). Esta operação está incluída no *software* fornecido pelo Docente da disciplina e é referida em bastante literatura, pelo que não irá ser aqui descrita. Contudo, devido ao uso de diferentes linguagens (nomeadamente, o *software* fornecido foi feito em Visual C++ e o programa elaborado em C), foi necessário proceder à respectiva adaptação quer à linguagem quer à forma como os dados se encontram representados.

3.3 Extracção de características

As características determinadas foram:

- pontos terminais (PT)
- entroncamentos (E)
- linhas rectas horizontais (RH)
- linhas rectas verticais (RV)
- linhas rectas diagonais com inclinação de 45º à direita (RD)
- linhas rectas diagonais com inclinação de 45º à esquerda (RE)

Estas características mostraram ser suficientes para efectuar um reconhecimento de algarismos.

Quer na extracção de PT quer de E, utilizou-se uma janela de 3x3 e um processo bastante simples.

No caso dos PT, determina-se que um dado pixel é um PT, caso na sua vizinhança só exista um pixel pertencente ao objecto a classificar; isto pode ser feito da seguinte forma: centrando o pixel a analisar numa janela 3x3 com valor 1, a soma de todos os pixels tem de ser igual a $(3^2 - 2)$ * (nível do fundo) + 2* nível do objecto. Neste caso, o nível do fundo é 255 – correspondente ao branco – e o nível do objecto é 0 – correspondente ao preto. Claro está que o pixel central tem de ter o mesmo nível que o nível do objecto – neste caso, 0.

Para determinar E, o processo utilizado foi o seguinte: usando uma janela de 3x3 com valor 1, diz-se que um dado pixel é um E se tiver o nível do objecto e se a soma dos pixels multiplicados pela janela centrada no pixel for igual a $(3^2 - 4)$ * nível do fundo + (3 – 1) * nível do objecto e se o valor do pixel for igual ao valor do objecto. Aqui pode ocorrer um problema: é que pontos vizinhos de um dado E também podem ser classificados, erradamente, como E; para que tal

não aconteça, verifica-se se um desses vizinhos já foi classificado como E; em caso afirmativo, então este pixel não é classificado como E; caso contrário, classifica-se este pixel como sendo um E. É óbvio que tal implica que quando um pixel for um E, este seja registado, para que se tenha a informação de quais os pixels já marcados como E.

Para reconhecer rectas foi usada uma janela 5x5, pois após uma análise aos algarismos disponíveis, concluiu-se que este seria o número que mais correctamente identificava todos os tipos de rectas.

Nos algoritmos implementados, um pixel é considerado como pertencente a uma recta, se 4 dos seus 25 vizinhos tiverem todos o mesmo nível que esse pixel, desde que todos os 5 pixels se encontrem na mesma direcção.

No algoritmo, pretendeu-se determinar qual o número de rectas existentes em cada imagem, sendo necessário, desse modo, registar os pixels pertencentes a uma dada recta; se existissem outros pixels vizinhos pertencentes a uma dada recta, então estes novos pixels não seriam registados. Tal como no caso dos entroncamentos, foi necessário fazer um registo das rectas existentes. Tal foi feito para todas as rectas.

3.4 Reconhecimento de algarismos

Para reconhecer os algarismos, atendendo às características de cada algarismo, usou-se a seguinte estratégia:

Se PT = 0
- Se E = 0, o algarismo é o **0**
- Caso contrário, o algarismo é o **8**

Se PT = 1
- Se o PT for mais alto que os E, então, o algarismo é o **6**
- Caso contrário
  - Se a primeira RV ou a última RH estiver abaixo do PT, então é um **0**
  - Caso contrário, é um **9**

Se PT = 2
- Se RH = 0, o algarismo é o **1**
- Caso contrário
  - Se houver E, então o algarismo é o **3**
  - Caso contrário
    - Se não houver qualquer RH acima e à direita do 1º PT, então é um **5**
    - Caso contrário, é um **2**

Se PT = 3
- Se RH = 0, o algarismo é o **1**
- Caso contrário
  - Se a 1ª RD estiver à esquerda e abaixo do 1º PT o algarismo é o **4**
  - Caso contrário
  - Se os dois últimos PT estiverem à esquerda do E
    - Se RE = 0, então o algarismo é o **7**
    - Caso contrário, é o 3

Se não existirem RD à esquerda e caso existam RH abaixo do 2º PT, então o algarismo é o **2**

Se PT = 4
O algarismo é o **3**

Caso não ocorra qualquer um destes casos, não possível reconhecer o algarismo.

4. Descrição do programa

O programa vai executar os algoritmos acima descritos de uma forma sequencial.
Começa por perguntar se deseja especificar um directório que contenha os ficheiros com os algarismos a reconhecer; tal é feito de forma a não ter se estar sempre a escrever todo o endereço para cada ficheiro. De igual modo, é desnecessário colocar a extensão do ficheiro, pois ele irá sempre procurar o ficheiro com o nome especificado, mas com extensão *.bmp.
De seguida pede-se o nome do ficheiro que se pretende ler.
O programa lê o ficheiro, efectua o adelgaçamento da imagem e o reconhecimento.
Por último é perguntado ao utilizador se este deseja continuar – eventualmente, com o reconhecimento de outro, ou do mesmo, algarismo. Em caso afirmativo, é pedido o nome do ficheiro (de notar que a especificação da escolha de directório só é feita uma vez).
Sempre que se introduz o nome de um ficheiro inexistente, alerta-se o utilizador para tal facto e pergunta-se se este pretende continuar com a execução do programa, ou não.

4.1 Considerações

Convém salientar que as imagens com os algarismos a reconhecer têm de ser armazenadas em ficheiros do tipo BITMAP com 8 bits por pixel, monocromático e que os algarismos devem ser escritos com uma cor escura sobre uma folha clara (preferencialmente, preto sobre o branco).
De igual modo, cada ficheiro deve conter um só algarismo; caso se pretenda reconhecer vários algarismos de uma só vez, tem de se completar o programa com um algoritmo de segmentação que isole cada um dos algarismos e só depois é que podem ser aplicados os algoritmos acima descritos.
Por último, convém referir que é necessário ter algum cuidado com a escrita dos números, apesar de se ter tentado elaborar algoritmos tão abrangentes quanto possível.
Estes dois últimos pontos foram tidos em consideração aquando da definição deste trabalho, pois a eliminação destas duas últimas tarefas seriam, eventualmente, alvo de um trabalho mais ambicioso do que este (por exemplo, como projecto de fim de curso), que tem de ser enquadrado com um trabalho para uma disciplina.

5. Conclusões

Este trabalho permitiu uma aprendizagem das técnicas usadas na construção de ficheiros de imagem (mais especificamente, os do tipo BITMAP).
De igual modo, permitiu desenvolver técnicas referidas na disciplina de Inspecção e Visão Industrial, pertencente ao Mestrado em Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto, no âmbito da qual foi realizado, bem como de outras disciplinas anteriores, aquando da frequência da Licenciatura em Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto.
Em anexo encontra-se o código, em C, do programa.

6. Bibliografia

- *Digital Image Processing*, R. Gonzalez, R. Woods, Addison-Wesley
- *Multimídia – Programação for Windows*, S. Rimmer, McGraw-Hill
- Apontamentos da disciplina *Inspecção e Visão Industrial* do Mestrado em Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto
- Apontamentos da disciplina *Processamento e Análise de Imagem* dao Licenciatura em Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto
- Várias páginas na Internet

Anexo

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

#define DimJanela 5

typedef unsigned char  BYTE;
typedef unsigned short WORD;
typedef unsigned long  DWORD;

int    tab[256]= { 2,0,0,1,0,2,1,1,0,2,1,1,1,2,1,1,0,2,2,2,2,2,2,2,1,2,
                   1,1,1,2,1,1,0,2,2,2,2,2,2,2,1,2,1,1,1,2,1,1,1,2,2,2,
                   2,2,2,2,1,2,1,1,1,2,1,1,0,2,2,2,2,2,2,2,2,2,2,2,2,2,
                   2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,
                   1,2,1,1,1,2,1,1,1,2,2,2,2,2,2,1,2,1,1,1,2,1,1,0,1,
                   1,1,2,2,1,1,2,2,1,1,2,2,1,1,2,2,2,2,2,2,2,2,2,2,1,1,
                   2,2,1,1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,
                   1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,1,1,2,2,1,1,2,2,1,1,
                   2,2,2,2,2,2,2,2,2,2,1,1,2,2,1,1,1,1,1,1,2,2,1,1,1,1,
                   1,1,1,1,1,1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,1 };

struct bmpheader {
        WORD  bfType;
  DWORD bfSize;
  WORD  bfReserved1;
  WORD  bfReserved2;
  DWORD bfOffBits;
 } bmph;

struct bmpinfoheader {
        DWORD biSize;
  DWORD biWidth;
  DWORD biHeigth;
  WORD  biPlanes;
  WORD  biBitCount;
  DWORD biCompression;
  DWORD biImageSize;
  DWORD biXPelsPerMeter;
  DWORD biYPelsPerMeter;
  DWORD biColorsUsed;
  DWORD biColorsImportant;
 } bmpih;

BYTE cores[4][256];
int matriz[500][500];
```

```
int xsize, ysize, cont;
int backLevel = 255, objLevel = 0;
BYTE linhas[500][500];
int terminos[5][2];
int entronc[10][2];
int rectasH[50][2];
int rectasV[50][2];
int rectasD[50][2];
int rectasE[50][2];

//------------------------------------------------------
int lerFicheiro(char nome[100])
{
 int i, j, desvio;
 FILE *fid;


 if ((fid=fopen(nome,"rb"))==NULL)
  {
   printf("Nao foi possivel abrir o ficheiro!");
   return(-1);
  }
 fseek(fid, 0, SEEK_SET);
 // leitura dos cabeçalhos e da tabela de cores
 fread(&bmph, 1, sizeof (bmpheader), fid);
 fread(&bmpih, 1, sizeof (bmpinfoheader), fid);

 // cálculo do desvio a introduzir no número de linhas e do número de
RGBQUAD's
 desvio = (bmpih.biImageSize / bmpih.biHeigth - bmpih.biWidth);
 j = (int)pow(2,bmpih.biBitCount);

 for (i=0; i<4; i++)
  fread(cores[i], j, sizeof (BYTE), fid);
 // leitura dos valores dos pixels
 for (i=0; i < bmpih.biHeigth; i++)
   fread(linhas[i], (bmpih.biWidth+desvio), sizeof (BYTE), fid);
 fclose(fid);

 // ordenação dos pixels
 for (i=0; i < bmpih.biHeigth; i++)
  for (j=0; j < bmpih.biWidth; j++)
   matriz[i][j]=(int)linhas[bmpih.biHeigth - 1 - i][j];
 return(0);
}

//------------------------------------------------------
int bindec(int x, int y)
{
 int  a = 1, b = 2, c = 4, d = 8, e = 16, f = 32, g = 64, h = 128, decimal;
```

```
int  q, r, s, t, u, v, w, z;

if ( y == 0 )
 {
  a = b = c = 0;
  if ( x == 0 )
    g = h = 0;
  if ( x == xsize-1 )
    d = e = 0;
 }
if ( y == ysize-1 )
 {
        e = f = g = 0;
        if ( x == 0 )
   a = h = 0;
        if ( x == xsize-1 )
   c = d = 0;
 }
if ( x == 0 )
 a = g = h = 0;
if ( x == xsize-1 )
 c = d = e = 0;

// outra coisa
if ( y == 0 || x == 0 )
 q = backLevel;
else
 q = matriz[x-1][y-1];
if (q != backLevel)
 q = 1;
else
 q = 0;

if ( y == 0  )
 r = backLevel;
else
 r = matriz[x][y-1];
if (r != backLevel)
 r = 1;
else
 r = 0;

if ( y == 0 || x == xsize-1 )
 s = backLevel;
else
 s = matriz[x+1][y-1];
if (s != backLevel)
 s = 1;
else
 s = 0;
```

```
if ( x == xsize-1 )
 t = backLevel;
else
 t = matriz[x+1][y];
if (t != backLevel)
 t=1;
else
 t=0;

if ( y == ysize-1 || x == xsize-1 )
 u = backLevel;
else
 u = matriz[x+1][y+1];
if (u != backLevel)
 u = 1;
else
 u = 0;

if ( y == ysize-1 )
 v = backLevel;
else
 v = matriz[x][y+1];
if
 (v != backLevel) v = 1;
else
 v = 0;

if ( y == ysize-1 || x == 0 )
 w = backLevel;
else
 w = matriz[x-1][y+1];
if (w != backLevel)
 w = 1;
else
 w = 0;

if (x == 0 )
 z = backLevel;
else
 z = matriz[x-1][y];
if (z != backLevel)
 z=1;
else
 z = 0;

decimal = a*q+b*r+c*s+d*t+e*u+f*v+g*w+h*z;

return(decimal);
}
```

```
//-----------------------------------------------------
int estrategia(int x, int y, int k)
{
 int  v;

 switch(k)
  {
   case 1 : {
                              if ( y == 0 || matriz[x][y-1] == backLevel )
              {
                                    v = bindec(x,y);
                                    if ( tab[v] == 1 )
                {
                                          matriz[x][y] = 128;
                                          cont ++;
                }
                               }
                                    break;
                                   }
   case 2 : {
                              if ( y == ysize-1 || matriz[x][y+1] == backLevel )
              {
                                    v = bindec(x,y);
                                    if ( tab[v] == 1 )
                {
                                          matriz[x][y] = 128;
                                          cont ++;
                }
                               }
                                    break;
                                   }
   case 3 : {
                              if ( x == xsize-1 || matriz[x+1][y] == backLevel )
              {
                                    v = bindec(x,y);
                                    if ( tab[v] == 1 )
                {
                                          matriz[x][y] = 128;
                                          cont ++;
                }
                               }
                                    break;
                                   }
          case 4 : {
                        if ( x == 0 || matriz[x-1][y] == backLevel )
            {
                                    v = bindec(x,y);
                                    if ( tab[v] == 1 )
```

```
            {
                                        matriz[x][y] = 128;
                                        cont ++;
            }
                        }
                             break;
                            }
          }
  return(0);
}

//-------------------------------------------------------
void limpeza()
{
 register int  x,y;

 for ( x = 0; x < xsize; x++ )
  {
   for ( y = 0; y < ysize; y++ )
          if ( matriz[x][y] == 128 )
     matriz[x][y] = backLevel;
  }
}

//-------------------------------------------------------
void ossos()
{
 register int x, y;
 int      k, flag;
 //int id = 0;

 xsize = (int)bmpih.biHeigth;
 ysize = (int)bmpih.biWidth;
 flag = 0;

 while ( flag != 4 )
  {
   flag = 0;
         for ( k = 1; k <= 4; k++ )
    {
     cont = 0;
           for ( x = 0; x < xsize; x++ )
      {
       for ( y = 0; y < ysize; y++ )
                    if ( matriz[x][y]  == objLevel )
         estrategia(x, y, k);

                   }
            if ( cont == 0) flag++;
            limpeza();
```

```
          }
  }
}

//-----------------------------------------------------
int terminacoes()
{
 int i, j, soma, term = 0;

 for (i = 2; i < (bmpih.biHeigth - 2); i++)
  for (j = 2; j < (bmpih.biWidth - 2); j++)
   {
/*      soma = matriz[i-2][j-2] + matriz[i-2][j-1] + matriz[i-2][j] + matriz[i-2][j+1] + matriz[i-2][j+2] +
                matriz[i-1][j-2] + matriz[i-1][j-1] + matriz[i-1][j] + matriz[i-1][j+1] + matriz[i-1][j+2] +
         matriz[i][j-2] + matriz[i][j-1] + matriz[i][j] + matriz[i][j+1] + matriz[i][j+2] +
                matriz[i+1][j-2] + matriz[i+1][j-1] + matriz[i+1][j] + matriz[i+1][j+1] + matriz[i+1][j+2] +
         matriz[i+2][j-2] + matriz[i+2][j-1] + matriz[i+2][j] + matriz[i+2][j+1] + matriz[i+2][j+2];
    if ((soma == 22*255) && (matriz[i][j] == 0))*/
    soma = matriz[i-1][j-1] + matriz[i-1][j] + matriz[i-1][j+1] +
         matriz[i][j-1] + matriz[i][j] + matriz[i][j+1] +
                matriz[i+1][j-1] + matriz[i+1][j] + matriz[i+1][j+1];
    if ((soma == 7*255) && (matriz[i][j] == 0))
     {
      terminos[term][0] = i;
      terminos[term][1] = j;
      term++;
     }
   }
 return(term);
}

//-----------------------------------------------------
int entroncamentos()
{
 int i, j, soma, ent=0, vizinho, h, v;

 for (i = 1; i < (bmpih.biHeigth - 1); i++)
  for (j = 1; j < (bmpih.biWidth - 1); j++)
   {
    soma = matriz[i-1][j-1] + matriz[i-1][j] + matriz[i-1][j+1] +
                matriz[i][j-1] + matriz[i][j] + matriz[i][j+1] +
         matriz[i+1][j-1] + matriz[i+1][j] + matriz[i+1][j+1];
    if ((soma == 5*255) && (matriz[i][j] == 0))
     {
      vizinho = 0;
```

```
      for (h = 0; h < 3; h++)
       for (v = 0; v < 3; v++)
        for (soma = 0; soma < ent; soma++)
          if ((entronc[soma][0] == (i - 1 + h)) && (entronc[soma][1] == (j - 1 +
v)))
            vizinho = 1;
      if (!vizinho)
      {
       entronc[ent][0] = i;
       entronc[ent][1] = j;
       ent++;
      }
     }
   }
 return(ent);
}


//-----------------------------------------------------
int rectasHor()
{
 int janela[DimJanela][DimJanela];
 int i, j, soma;
 int h, v;
 int rectH = 0, vizinho;

 // definição de uma janela para detectar rectas horizontais
 for (i = 0; i < DimJanela / 2; i++)
  for (j = 0; j < DimJanela; j++)
   janela[i][j] = -1;
 for (i= 0 ; i < DimJanela; i++)
  janela[DimJanela / 2][i] = DimJanela - 1;
 for (i = (DimJanela / 2 + 1); i < DimJanela; i++)
  for (j = 0; j < DimJanela; j++)
   janela[i][j] = 1;

 // procura das rectas
 for (i = DimJanela / 2; i < (bmpih.biHeigth - DimJanela / 2); i++)
  for (j = DimJanela / 2; j < (bmpih.biWidth - DimJanela / 2); j++)
   {
    soma = 0;
    for (h = 0; h < DimJanela; h++)
     for (v = 0; v < DimJanela; v++)
      soma += (matriz[(int)(i - DimJanela / 2 + h)][(int)(j  - DimJanela / 2 +
v)]*janela[h][v]);
    if (soma == 0)
     {
       vizinho = 0;
       for (v = 0; v < DimJanela; v++)
        if (matriz[i][(int)(j - DimJanela / 2 + v)] != 0)
```

```
          {
           vizinho = 1;
           break;
          }
        if (!vizinho)
         {
          v = j;
          if (rectH != 0)
           {
            vizinho = 0;
            while(matriz[i][v] == 0)
             {
              v--;
              for (h = 0; h < rectH; h++)
               if ((i == rectasH[h][0]) && (v == rectasH[h][1]))
                {
                 vizinho = 1;
                 break;
                }
              if (vizinho)
               break;
             }
            if (!vizinho)
             {
              rectasH[rectH][0] = i;
              rectasH[rectH][1] = j;
              rectH++;
             }
           }
          else
           {
            rectasH[rectH][0] = i;
            rectasH[rectH][1] = j;
            rectH++;
           }
         }
      } // if (soma == ... )
    }

 return(rectH);
}

//-----------------------------------------------------
int rectasVer()
{
 int janela[DimJanela][DimJanela];
 int i, j, soma;
 int h, v;
 int rectV = 0, vizinho;
```

```
// definição de uma janela para detectar rectas verticais
for (i = 0; i < DimJanela; i++)
 for (j = 0; j < DimJanela / 2; j++)
  janela[i][j] = -1;
for (i = 0; i < DimJanela; i++)
 janela[i][DimJanela / 2] = DimJanela - 1;
for (i = 0; i < DimJanela; i++)
 for (j = (DimJanela / 2 + 1); j < DimJanela; j++)
  janela[i][j] = +1;

// procura das rectas
for (i = DimJanela / 2; i < (bmpih.biHeigth - DimJanela / 2); i++)
 for (j = DimJanela / 2; j < (bmpih.biWidth - DimJanela / 2); j++)
  {
   soma = 0;
   for (h = 0; h < DimJanela; h++)
    for (v = 0; v < DimJanela; v++)
     soma += (matriz[(int)(i - DimJanela / 2 + h)][(int)(j  - DimJanela / 2 +
v)]*janela[h][v]);
   if (soma == 0)
    {
     vizinho = 0;
     for (h = 0; h < DimJanela; h++)
      if (matriz[(int)(i - DimJanela / 2 + v)][j] != 0)
       {
        vizinho = 1;
        break;
       }
     if (!vizinho)
      {
       h = i;
       if (rectV != 0)
        {
         vizinho = 0;
         while(matriz[h][j] == 0)
          {
           h--;
           for (v = 0; v < rectV; v++)
            if ((h == rectasV[v][0]) && (j == rectasV[v][1]))
             {
              vizinho = 1;
              break;
             }
            if (vizinho)
             break;
          }
         if (!vizinho)
          {
           rectasV[rectV][0] = i;
           rectasV[rectV][1] = j;
```

```
            rectV++;
           }
          }
         else
          {
           rectasV[rectV][0] = i;
           rectasV[rectV][1] = j;
           rectV++;
          }
        }
    } // if (soma == ... )
   }

 return(rectV);
}

//------------------------------------------------------
int rectasDir()
{
 int janela[DimJanela][DimJanela];
 int i, j, soma;
 int h, v, aux;
 int rectD = 0, vizinho;


 // definição de uma janela para detectar rectas diagonais para a direita
(/)
 for (i = 0; i < DimJanela; i++)
  for (j = 0; j < DimJanela; j++)
   if (i == (DimJanela - 1 - j))
    janela[i][j] = DimJanela - 1;
   else
    {
     if (j > (DimJanela - 1 - i))
      janela[i][j] = -1;
     else
      janela[i][j] = 1;
    }

 // procura das rectas
 for (i = DimJanela / 2; i < (bmpih.biHeigth - DimJanela / 2); i++)
  for (j = DimJanela / 2; j < ((bmpih.biWidth) - DimJanela / 2); j++)
   {
    soma = 0;
    for (h = 0; h < DimJanela; h++)
     for (v = 0; v < DimJanela; v++)
       soma += (matriz[(int)(i - DimJanela / 2 + h)][(int)(j  - DimJanela / 2 +
v)]*janela[h][v]);
    if (soma == 0)
     {
```

```
        vizinho = 0;
        for (h = 0; h < DimJanela; h++)
           if (matriz[(int)(i - DimJanela / 2 + h)][(int)(j + DimJanela / 2 - h)] !=
0)
            {
             vizinho = 1;
              break;
            }
        if (!vizinho)
         {
          h = i;
          v = j;
          if (rectD != 0)
           {
            vizinho = 0;
            while(matriz[h][v] == 0)
             {
              h--;
              v++;
              for (aux = 0; aux < rectD; aux++)
               if ((h == rectasD[aux][0]) && (v == rectasD[aux][1]))
                {
                 vizinho = 1;
                 break;
                }
               if (vizinho)
                break;
             }
            if (!vizinho)
             {
              rectasD[rectD][0] = i;
              rectasD[rectD][1] = j;
              rectD++;
             }
           }
          else
           {
            rectasD[rectD][0] = i;
            rectasD[rectD][1] = j;
            rectD++;
           }
         }
      } // if (soma == ... )
    }
  return(rectD);
}

//-----------------------------------------------------
int rectasEsq()
{
```

```
 int janela[DimJanela][DimJanela];
 int i, j, soma;
 int h, v, aux;
 int rectE = 0, vizinho;

 // definição de uma janela para detectar rectas diagonais para a
esquerda (\)
 for (i = 0; i < DimJanela; i++)
  for (j = 0; j < DimJanela; j++)
   if (i == j)
    janela[i][j] = DimJanela - 1;
   else
    {
     if (j > i)
      janela[i][j] = -1;
     else
      janela[i][j] = 1;
    }

 // procura das rectas
 for (i = DimJanela / 2; i < (bmpih.biHeigth - DimJanela / 2); i++)
  for (j = DimJanela / 2; j < ((bmpih.biWidth) - DimJanela / 2); j++)
   {
    soma = 0;
    for (h = 0; h < DimJanela; h++)
     for (v = 0; v < DimJanela; v++)
      soma += (matriz[(int)(i - DimJanela / 2 + h)][(int)(j  - DimJanela / 2 +
v)]*janela[h][v]);
    if (soma == 0)
     {
      vizinho = 0;
      for (h = 0; h < DimJanela; h++)
          if (matriz[(int)(i - DimJanela / 2 + h)][(int)(j - DimJanela / 2 + h)] !=
0)
           {
            vizinho = 1;
            break;
           }
      if (!vizinho)
       {
        h = i;
        v = j;
        if (rectE != 0)
         {
          vizinho = 0;
          while(matriz[h][v] == 0)
           {
            h--;
            v--;
            for (aux = 0; aux < rectE; aux++)
```

```
            if ((h == rectasE[aux][0]) && (v == rectasE[aux][1]))
             {
              vizinho = 1;
              break;
             }
            if (vizinho)
             break;
           }
          if (!vizinho)
           {
            rectasE[rectE][0] = i;
            rectasE[rectE][1] = j;
            rectE++;
           }
         }
        else
         {
          rectasE[rectE][0] = i;
          rectasE[rectE][1] = j;
          rectE++;
         }
       }
    } // if (soma == ... )
  }
 return(rectE);
}

//-------------------------------------------------------
int Reconhecimento()
{
 int i, ha, nTerm, nEnt, nRectH, nRectD, nRectE;

 nTerm = terminacoes();
 nEnt = entroncamentos();
 nRectH = rectasHor();
 nRectD = rectasDir();
 nRectE = rectasEsq();

 // 0 terminações
 if (nTerm == 0)
  {
   if (!nEnt)
    return(0);
   else
    return(8);
  }

 // 1 terminação
 if (nTerm == 1)
  {
```

```
  if (terminos[0][0] < entronc[0][0])
   return(6);
  else
   if ((rectasH[nRectH - 1][0] > terminos[0][0]) || (rectasV[0][0] >
terminos[0][0]))
    return(0);
   else
    return(9);
 }

 // 2 terminações
 if (nTerm == 2)
  {
  if (nRectH == 0)
   return(1);
  else
  {
   if (nEnt != 0)
    return(3);
   else
     ha = 0;
     for (i = 0; i < nRectH; i++)
      if ((rectasH[i][0] < terminos[0][0]) && (rectasH[i][1] > terminos[0][1]))
       ha = 1;
     if (!ha)
      return(5);
     else
      return(2);
  }
 }

 // 3 terminações
 if (nTerm == 3)
  {
  if (nRectH == 0)
   return(1);
  else
   {
    if (nRectD != 0)
     if ((rectasD[0][0] < entronc[0][0]) && (rectasD[0][1] < terminos[0][1]))
      return(4);

    if ((terminos[1][1] < entronc[0][1]) && (terminos[2][1] < entronc[0][1]))
     if (nRectE == 0)
      return(7);
     else
      return(3);

    if ((nRectD != 0) && (nRectH != 0))
```

```
                if ((terminos[1][1] < rectasD[nRectD - 1][1]) &&
(terminos[1][0] < rectasH[nRectH - 1][0]))
      return(2);
    }
  }

 // 4 terminações
 if (nTerm == 4)
  return(3);

 return(-1);
}

//-----------------------------------------------------
void main()
{
 int digitoRec, cond;
 char nome[100], fich[10];

 cond = 1;
 printf("Deseja especificar o directorio em que se encontram os ficheiros
(s/n)? ");
 scanf("%s", fich);
 if (strcmp(fich, "s") == 0)
  {
   printf("\nPor favor, introduza o nome do directorio: ");
   scanf("%s", nome);
   strcat(nome, "\\");
  }
 else
  strcpy(nome, "");
 while (cond == 1)
 {
  printf("\nPor favor, introduza o nome do ficheiro BITMAP (sem a
extensao) a reconhecer:\n");
  scanf("%s", fich);
  strcat(nome, fich);
  strcat(nome,".bmp");
  if ((lerFicheiro(nome)) < 0)
   printf("\nO ficheiro %s nao existe!\n", fich);
  else
   {
    ossos();
    if ((digitoRec = Reconhecimento()) < 0)
     printf("\nNao foi reconhecido qualquer algarismo!\n");
    else
     printf("\nO algarismo reconhecido foi o %d!\n", digitoRec);
   }
  printf("\n\nDeseja continuar (s/n)?");
  scanf("%s", nome);
```

```
  digitoRec = strcmp(nome, "n");
  if (digitoRec == 0)
   cond = 0;
  clrscr();
 }
}
```