Trabalho Prático 2
Relatório de Desenvolvimento

Mário Ulisses Pires Araújo Costa (43175)
Nuno André Veloso Matias (44229)
Vasco Miguel Borges de Almeida Ferreira (43207)

10 de Abril de 2006

Resumo

Este trabalho foi implementado no paradigma imperativo, em liguagem C.
O objectivo é conceber uma aplicação cujo intuito seja possibilitar o utilizador a concepção de uma série de operações gráficas de desenho bem como de gestão dos mesmos desenhos.

Conteúdo

1 Introdução

Este trabalho foi realizado no âmbito da disciplina de Paradigmas da Programação 2, da Licenciatura em Engenharia de Sistemas e Informática da Universidade do Minho.
O objectivo é conceber uma aplicação cujo intuito seja possibilitar o utilizador a conceber uma série de operações grficas de desenho bem como de gestão dos mesmos desenhos.
Este relatório contem as diferentes fases do desenvolvimento do programa, a saber: análise do problema, escolha/descoberta do algoritmo, implementação, testes e documentação.

2 Análise e Especificação

2.1 Descrição informal do problema

Partindo do que nos foi indicado pelo enunciado,o lápis será substituído por comandos fornecidos pelo utilizador (descreve-se mais à frente esta interacção) e o papel será substituído por uma matriz de pontos que o utilizador poderá dimensionar de início.
Matriz essa qeu será posteriormente gravada em formato de imagem para um ficheiro do tipo *PBM*, se o utilizador assim pretender.

2.2 Especificação dos Requisitos

Para levar a cabo este trabalho, foram necessários conhecimentos da linguagem C, conhecimentos de matemática, do formato de imagens *PBM*, assim como o domínio de LaTeX 2_ε e alguns dos seus pacotes para elaborar este relatório.

2.2.1 Dados

O desenho para o qual será gravada a imagem depois de ser alterada pela *aplicação DESENHO* estará em formato *PBM* que tem o seguinte formato:

- Na primeira linha, grava-se um identificador do formato.
- Na segunda linha, coloca-se um comentário descritivo.
- Na terceira linha, são gravados dois inteiros, o primeiro corresponde ao número de colunas da imagem e o segundo ao número de linhas.
- As restantes linhas do ficheiro correspondem às linhas do desenho. Cada linha do desenho é gravada numa linha do ficheiro. Cada ponto do desenho é gravado como um inteiro e é separado do seguinte por um espaço. Cada linha do ficheiro é separada da seguinte por uma mudança de linha (carácter '\n').

Exemplo O seguinte texto;

```
P1
# Imagem PBM explicativa do formato
40 15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

. . . iriá originar a seguinte imagem:

Figura 1: Um exemplo de uma imagem no formato PBM

2.2.2 Pedidos

Foi-nos pedido a criação de uma aplicação que deva suportar uma série de operações gráficas de desenho que serão realizadas sobre a matriz de pontos bem como as de gestão.
Listam-se a seguir, com maior detalhe, essas operações desejadas:

Operações de gestão

Dim linhas, colunas Este comando permite estabelecer as dimensões da área de desenho.
O valor linhas é um inteiro que especifica o número de linhas da matriz e o valor colunas, o inteiro que especifica o número de colunas da matriz.
Este comando deverá ser usado uma única vez no início do desenho.

SetMarker marca Este comando permite definir o carácter que será usado no traçado do desenho.
O valor marca corresponde ao carácter que se quer usar para traçar as linhas.
Este comando pode ser executado em qualquer altura. Se fôr executado a meio de um desenho permite trocar o carácter que se está a usar para desenhar.
Por omissão e no início do programa a marca está definida como sendo o ó minúsculo.

Load desenho Este comando permite carregar um desenho previamente gravado num ficheiro.
O valor desenho corresponde ao nome do ficheiro onde está gravado o desenho.
Este comando foi fornecido pelo docente.

Save desenho Este comando permite gravar um desenho num ficheiro.
O valor desenho corresponde ao nome do ficheiro onde será gravado o desenho.
Este comando foi fornecido pelo docente.

Operações gráficas

- Point x,y
  Este comando marca um ponto nas coordenadas x,y, ou seja, naquela posição da matriz deverá ser colocado o carácter definido como a marca actual.

- Line x_1,y_1,x_2,y_2
  Este comando traça uma linha marcando os pontos que se aproximam da linha recta entre as coordenadas x_1,y_1 e x_2,y_2.

- Rect x_1,y_1,x_2,y_2
  Este comando traça um rectângulo assumindo os pontos x_1,y_1 e x_2,y_2 como os limites de uma das suas diagonais.

- PolyLine x_1,y_1,x_2,y_2,...,x_n,y_n
  Este comando traça uma linha unindo os pontos cujas coordenadas foram fornecidas. Desta forma permite abreviar alguns traçados.

  Exemplo PolyLine x_1,y_1,x_2,y_2,x_3,y_3,x_4,y_4 é equivalente a:
  Line x_1,y_1,x_2,y_2
  Line x_2,y_2,x_3,y_3
  Line x_3,y_3,x_4,y_4

- Circ x,y,r
  Este comando permite traçar uma circunferência com centro em x,y e de raio r.

- Elip x_1,y_1,x_2,y_2
  Este comando permite traçar uma elipse com focos em x_1,y_1 e x_2,y_2 respectivamente.

3 Concepção/Desenho da Resolução

3.1 Estruturas de Dados

Para a realização deste trabalho recorremos ao uso dos seguintes tipos de dados:

Apontadores de Caracteres para gurdar os valores inseridos pelo utilizador quando invoca um comando de desenho;

Inteiros para os parametros das funções de desenho;

Array's de inteiros para a função polyLine tratar

3.2 Estrutura da Aplicação

3.2.1 O menu inicial da aplicação

Este menu irá pôr ao dispôr do utilizador um conjunto de 3 operações num total de 4 escolhas:

1 - Cria um desenho novo
2 - Abrir um desenho guardado
3 - Guardar um desenho criado
3 - Set marca

0 - Sair

A opção *3 - Guardar um desenho criado*, convém, apenas ser usada no caso do utilizador ter criado um desenho novo, pois o programa parte do principio que se abriu um desenho já guardado não o irá guardar com o mesmo nome, para não perder o antigo, daí apenas ser útil usar esta opção quando tiver criado um desenho novo.
Esta opção, no caso de ser usada depois de abrir um desenho previamente guardado, manda o desenho alterado para uma imagem como nome *out.pbm*, para evitar perdas de dados.
A opção *4 - Set marca* permite ao utilizador inserir a marca que pretende para visualizar o traçado do desenho.

3.2.2 Leitura dos comandos de desenho

Para lermos os comandos que o utilizador insere optamos por um *prompt* vazio, apenas com a indicação de como sair;

Prima 's' para voltar

Aqui o utilizador poderá inserir os seguintes, e só os seguintes comandos serão aceites;

- Point x, y
- PolyLine $x_1, y_2, \ldots, x_n, y_n$
- Line x_1, y_1, x_2, y_2

- Rect x_1, y_1, x_2, y_2
- Circ x, y, r
- Elip x_1, y_1, x_2, y_2
- **s** no caso de querer sair deste *prompt*

A leitura do numero de argumentos dos comandos acima listados é tratada pela função *opcoes* e expressa na seguinte tabela:

Comando	**nr de arg's**
Point	2
PolyLine	$2n, \forall n \in \mathbb{N}$
Line	4
Rect	4
Circ	3
Elip	4

No caso do utilizador inserir um número superior ao desejado por cada comando, apenas serão enviados os primeiros necessários para usar esse comando.

Exemplo se o utilizador inseriu PolyLine $1, 2, 3, 4, 5$
Apenas serão enviados para a função PolyLine os 4 primeiros números, pois esta apenas aceita uma quantidade de argumentos cujo seu somatório seja números pares (na forma $2n$).

3.3 Algoritmos

> *Parametric equations are a set of equations that express a set of quantities as explicit functions of a number of independent variables, known as "parameters."*

3.3.1 Algoritmos Matemáticos

Para a resolução deste trabalho usamos os seguintes:

Equações de parametrização do círculo;
Sendo r o raio da circunferência e α um qualquer angulo:

$$x = r * cos(\alpha) \tag{1}$$

$$y = r * sin(\alpha) \tag{2}$$

Equações de parametrização da elipse;
Sendo (x_2, y_2) o segundo foco da elipse e α um qualquer angulo:

$$x = x_2 * cos(\alpha) \tag{3}$$

$$y = y_2 * sin(\alpha) \tag{4}$$

Equações de parametrização da recta;
Sendo m o declive da recta, dado pela equação (6), e b dado pela equação (7), temos:

$$y = m * x + b \tag{5}$$

$$m = \frac{y_1 - y_2}{x_1 - x_2} \tag{6}$$

$$b = y - m * x \tag{7}$$

3.3.2 Comando Circ

Tal como ja vimos através das equações (1) e (2) temos aí um bom algoritmo que, ao que nos pareceu, fácil de implementar, optamos por fazer da seguinte maneira:

```
/*
  Propriedades para desenhar um circulo:
    a)  'l' e 'c'  SEMPRE >= que 'r'
    b1) 'l'+'r'    SEMPRE <= MAXLINHAS
    b2) 'c'+'r'+1  SEMPRE <= MAXCOLUNAS
*/

int circ(int l , int c , int r , DESENHO d)
{
  int a,ret=0,linhas=d[0][0],colunas=d[0][1];

  if((l>=r && c>=r) || (l+r<=linhas && c+r+1<=colunas))
  {
        for(a=0;a<360*r;a++)
        {
                int px = r * cos(a) , py = r * sin(a) , yy = py+c , xx = px+l;

                d[xx][yy]=marca;
        }
        ret=1;
  }
  return ret;
}
```

3.3.3 Comando Elip

Tal como para o circulo através das equações (3) e (4) é fácil implementar uma solução:

```
/*
  Propriedades para desenhar uma elipse:
    a1)  'x1' e 'y1' SEMPRE >= x2
    b1) 'x1'+'x2'    SEMPRE <= MAXLINHAS
    b2) 'y1'+'y2'+1  SEMPRE <= MAXCOLUNAS
*/

int elip(int x1, int y1, int x2, int y2, DESENHO d)
{
```

```
10    int a,ret=0,linhas=d[0][0],colunas=d[0][1];
11
12    if((x1>=x2 && y1>=y2) || (x1+x2<=linhas && y1+y2+1<=colunas))
13    {
14          for(a=0;a<360*(x2+y2);a++)
15          {
16                  int px = x2 * cos(a) , py = y2 * sin(a) , yy = py+y1 , xx = px+x1;
17
18                  d[xx][yy]=marca;
19          }
20          ret=1;
21    }
22    return ret;
23 }
```

3.3.4 Comando PolyLine

O algoritmo escolhido para resolver a leitura de n variáveis pelo comando PolyLine foi;
Despois de termos a string dos n argumentos armazenado na variável *pontos*, esta entra num ciclo que só irá parar quando esta seja igial a *NULL*.
Usamos a função strchr que recebe dois argumentos, o primeiro uma string s e o segundo um caracter c e que devolve um apontador para a primeira posição da string s onde ocorra o caracter c, incrementamos uma unidade a este resultado pois desta forma o apontador pontos ficará a apontar para a primeira ocorrencia de um número na string pontos (já alterada pela strchr).
Depois, através do comando sscanf lemos tudo ate a proxima virgula e mandamos esse valor (float) para um array do mesmo tipo, um exemplo:

```
1  pontos = "10,20,30,40,50,60"
2  ————————→sscanf(pontos,"%[^,]", acc)   // a1) guardar a string ate a virgla de 'pontos' em acc
3                                          //     acc = "10"
4  ————————→mtz[0] = 10                   // b1) guardar no array como float
5  pontos = strchr(pontos,',')+1          // c1) pontos = "20,30,40,50,60"
6
7  ————————→sscanf(pontos,"%[^,]", acc)   // a2) acc = "20"
8  ————————→mtz[1] = 20                   // b2) guardar no array como float
9  pontos = strchr(pontos,',')+1          // c2) pontos = "30,40,50,60"
10
11 ————————→sscanf(pontos,"%[^,]", acc)   // a3) acc = "30"
12 ————————→mtz[2] = 30                   // b3) guardar no array como float
13 pontos = strchr(pontos,',')+1          // c3) pontos = "40,50,60"
14
15 ————————→sscanf(pontos,"%[^,]", acc)   // a4) acc = "40"
16 ————————→mtz[3] = 40                   // b4) guardar no array como float
17 pontos = strchr(pontos,',')+1          // c4) pontos = "50,60"
18
19 ————————→sscanf(pontos,"%[^,]", acc)   // a5) acc = "50"
20 ————————→mtz[4] = 50                   // b5) guardar no array como float
21 pontos = strchr(pontos,',')+1          // c5) pontos = "60"
22
23 ————————→sscanf(pontos,"%[^,]", acc)   // a6) acc = "60"
24 ————————→mtz[5] = 60                   // b6) guardar no array como float
25 pontos = strchr(pontos,',')+1          // c6) pontos = ""
26 //saimos fora do ciclo...
```

Em C teriamos o seguinte excerto de código:

```
int b;
      if(strcmp(comando,"PolyLine")==0)
      {
              float mtz[100];

              for(b=0 ; pontos-1!=NULL ; b++ , pontos = strchr(pontos,',')+1)
              {
                      char *acc = (char *)calloc(100,sizeof(char));

                      sscanf (pontos,"%[^,]", acc);
                      mtz[b] = atoi(acc);
              }
              polyLine(mtz,b-1,d1);
              comando=""; // comando = 'PolyLine' contem 'Line'
      }
```

Despois dos valores armazenados no vector de float's mtz a função polyLine recebe-o, com o $b = \text{comprimento}(mtz) - 1$ e a matriz onde se quer desenhar. A função polyLine tem um contador i que incrementa de 2 em 2, o que nos permite avançar de 2 em 2 posições no array mtz que é comparado, a cada iteração do ciclo, ao $b - 2$, um exemplo:

Imaginemos que temos os seguintes dados;

```
DESENHO d1;

float mtz = [10,20,30,40,50,60,70,80,90,100];

int b = length(mtz) − 1;
```

. . . e que invocamos a função da seguinte forma;

$\hookrightarrow polyLine(mtz, b, d1)$

. . . iriamos ter o seguinte algoritmo:

```
// b = 9

(i=0) < (b-2=7)                                          // verdade -> entramos no ciclo
————————→line(mtz[i],mtz[i+1],mtz[i+2],mtz[i+3],d1); // mtz[0]=10
                                                         // mtz[1]=20
                                                         // mtz[2]=30
                                                         // mtz[3]=40
                                                         // line(10,20,30,40,d1);

(i=2) < (b-2=7)                                          // verdade -> entramos no ciclo
————————→line(mtz[i],mtz[i+1],mtz[i+2],mtz[i+3],d1); // mtz[2]=30
                                                         // mtz[3]=40
                                                         // mtz[4]=50
                                                         // mtz[5]=60
                                                         // line(30,40,50,60,d1);

(i=4) < (b-2=7)                                          // verdade -> entramos no ciclo
————————→line(mtz[i],mtz[i+1],mtz[i+2],mtz[i+3],d1)  // mtz[4]=50
                                                         // mtz[5]=60
                                                         // mtz[6]=70
                                                         // mtz[7]=80
                                                         // line(50,60,70,80,d1);
```

```
23
24 (i=6) < (b-2=7)                                          // verdade -> entramos no ciclo
25 ————————→line(mtz[i],mtz[i+1],mtz[i+2],mtz[i+3],d1);     // mtz[6]=70
26                                                          // mtz[7]=80
27                                                          // mtz[8]=90
28                                                          // mtz[9]=100
29                                                          // line(70,80,90,100,d1);
30
31 (i=8) < (b-2=7)                                          // falso   -> saimos ciclo
```

Como podemos ver, acabamos por executar as seguintes linhas:

```
line(10,20,30,40);

line(30,40,50,60);

line(50,60,70,80);

line(70,80,90,100);
```

. . . como era pretendido.

Em C teriamos o seguinte código:

```
1 void polyLine(float a[], int b, DESENHO d)
2 {
3         int i;
4
5         for(i=0;i<b-2;i+=2)
6                 line(a[i],a[i+1],a[i+2],a[i+3],d);
7 }
```

4 Codificação e Testes

4.1 Problemas de Implementação

Um dos grandes problemas em fazer umprograma são as validações, nós tentamos ver todos os casos possíveis e validar todos esses casos.

4.2 Alternativas e Decisões

Quando o utilizador usa o *prompt* vazio tem liberdade total para escrever o que quiser. Por isso para "filtrar"apenas o que queremos usamos a função predefinida strstr e assim procuramos a ocorrência do que queriamos conforme o que o utilizador escreveu.
Caso não haja uma relação entre o que ele escreveu e o que pretendemos não se faz nada e volta-se a mostrar o mesmo *prompt*.

Para validar o número de parametros que eram introduzidos pelo utilizador no mesmo *prompt* ja tivemos que recorrer a uma solução nossa, que mostramos asseguir:

```c
int valida(char *op, int n)
{
  int i,ret=0;

  for(i=0;op-1!=NULL;i++,op = strchr(op,',')+1);
  if (i==n) ret=1;

  return ret;
}
```

Esta solução recebe os parametros que o utilizador escreveu, numa string, mais o numero de parametros que se esperam que la estejam. Ao começar o i em 0 permite-nos contar o numero de parametros introduzidos, que depois do ciclo acabar sao comparados ao n (numero de parametros desejados).
Se coincidir o programa devolve 1 se não devolve 0

4.3 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos (valores introduzidos) e os respectivos resultados obtidos:

Teste 1 *prompt*

```
> PolyLine 1,1,8,10,8,20,20,30,10,40
10000000000000000000000000000000000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000000000000000000000000000000000
00100000000000000000000000000000000000000000000000000000000000000000000000000000
00010000000000000000000000000000000000000000000000000000000000000000000000000000
00000100000000000000000000000000000000000000000000000000000000000000000000000000
00000010000000000000000000000000000000000000000000000000000000000000000000000000
00000001000000000000000000000000000000000000000000000000000000000000000000000000
00000000011111111111000000000000000000000000000000000000000000000000000000000000
00000000000000000001000000000000000000000000000000000000000000000000000000000000
00000000000000000000100000000000000000010000000000000000000000000000000000000000
00000000000000000000010000000000000000100000000000000000000000000000000000000000
00000000000000000000001000000000000001000000000000000000000000000000000000000000
00000000000000000000000100000000000010000000000000000000000000000000000000000000
00000000000000000000000010000000000100000000000000000000000000000000000000000000
00000000000000000000000010000000001000000000000000000000000000000000000000000000
00000000000000000000000001000000010000000000000000000000000000000000000000000000
00000000000000000000000000100000100000000000000000000000000000000000000000000000
00000000000000000000000000010001000000000000000000000000000000000000000000000000
00000000000000000000000000001010000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
        Prima 's' para voltar
```

. . . iriá originar a seguinte imagem:

Teste 2 *prompt*

Figura 2: Teste 1 do programa de DESENHO

```
> Circ 15,40,7

00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000011111110000000000000000000000000000000000000
00000000000000000000000000000000000110000011000000000000000000000000000000000000
00000000000000000000000000000000001100000001100000000000000000000000000000000000
00000000000000000000000000000000011000000000110000000000000000000000000000000000
00000000000000000000000000000000010000000000010000000000000000000000000000000000
00000000000000000000000000000000010000000000010000000000000000000000000000000000
00000000000000000000000000000000010000000000010000000000000000000000000000000000
00000000000000000000000000000000010000000000010000000000000000000000000000000000
00000000000000000000000000000000010000000000010000000000000000000000000000000000
00000000000000000000000000000000011000000000110000000000000000000000000000000000
00000000000000000000000000000000001100000001100000000000000000000000000000000000
00000000000000000000000000000000000110000011000000000000000000000000000000000000
00000000000000000000000000000000000011111110000000000000000000000000000000000000
00000000000000000000000000000000000000010000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
        Prima ’s’ para voltar
```

... iriá originar a seguinte imagem:

Figura 3: Teste 2 do programa de DESENHO

Teste 3 *prompt*

```
> Elip 15,15,7,11
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000011111111111000000000000000000000000000000000000000000000000000000000000
00000001110000000001110000000000000000000000000000000000000000000000000000000000
00000011000000000000011000000000000000000000000000000000000000000000000000000000
00000100000000000000000100000000000000000000000000000000000000000000000000000000
```

```
14 00001100000000000000000011000000000000000000000000000000000000000000000000000000000000
15 00001000000000000000000010000000000000000000000000000000000000000000000000000000000000
16 00001000000000000000000010000000000000000000000000000000000000000000000000000000000000
17 00001000000000000000000010000000000000000000000000000000000000000000000000000000000000
18 00001100000000000000000011000000000000000000000000000000000000000000000000000000000000
19 00000100000000000000000010000000000000000000000000000000000000000000000000000000000000
20 00000011000000000000001100000000000000000000000000000000000000000000000000000000000000
21 00000001110000000001110000000000000000000000000000000000000000000000000000000000000000
22 00000000011111111111000000000000000000000000000000000000000000000000000000000000000000
23 00000000000001000000000000000000000000000000000000000000000000000000000000000000000000
24 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
25 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
26 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
27          Prima 's' para voltar
```

... iriá originar a seguinte imagem:

Figura 4: Teste 2 do programa de DESENHO

5 Conclusão

Todos os requesitos desejados foram concretizados com sucesso.

Este trabalho permitiu-nos desenvolver uma ferramenta que julgamos ser bastante útil até mesmo em situações reais.

Após a conclusão deste relatório, os nossos conhecimentos em LaTeX 2_ε foram reforçados, o que representa outro grande objectivo alcançado.

A Código do Programa

Lista-se a seguir o código C do programa que foi desenvolvido.

```
/* ------------------------------------------------------
   Programa DESENHO que utiliza as funcoes de
   leitura e escrita de desenhos

   2006-03-10: created by
       Mario Ulisses Pires Araujo Costa        (43175)
       Nuno Andre Veloso Matias                (44229)
       Vasco Miguel Borges de Almeida Ferreira (43207)
   ------------------------------------------------------ */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "myio.c"

/* ------------------------------------------------------------
   Declaracao de funcoes
   ------------------------------------------------------------ */

int circ(int l , int c , int r , DESENHO d);
int elip(int x1, int y1, int x2, int y2, DESENHO d);
void line(float x1, float y1, float x2, float y2, DESENHO d);
void point(int x, int y, DESENHO d);
void rect(int x1, int y1, int x2, int y2, DESENHO d);
void polyLine(float a[], int b, DESENHO d);
void menu(DESENHO d1);
void opcoes (DESENHO d1);
void criaFile(int l,int c,char fnome[]);
int valida(char *op, int n);
void setMarca(int n);

/* ------------------------------------------------------------
   Declaracao de variaveis globais
   ------------------------------------------------------------ */

int marca=1;

/* ------------------------------------------------------------
   Inicio do codigo da aplicacao DESENHO
   ------------------------------------------------------------ */

int main()
{
  DESENHO d1;

  menu(d1);

  return 0;
}

void menu(DESENHO d1)
{
  char *opcao=(char *)calloc(100,sizeof(char)), *marca=(char *)calloc(2,sizeof(char));

  system("clear");
  puts("\n\n\n");
  puts("\t1 - Cria um desenho novo");
  puts("\t2 - Abrir um desenho guardado");
  puts("\t3 - Guardar um desenho criado");
  puts("\t4 - Set marca\n");
  printf("\t0 - Sair\n\n> ");
```

```
63    scanf("%s",opcao);
64
65    if(strcmp(opcao,"1")==0)
66    {
67          char *colunas=(char *)calloc(100,sizeof(char));
68          char *linhas=(char *)calloc(100,sizeof(char));
69          char *fnome=(char *)calloc(100,sizeof(char));
70
71          printf("\tIntroduza o numero de colunas: ");
72          scanf("%s",colunas);
73          printf("\tIntroduza o numero de linhas: ");
74          scanf("%s",linhas);
75          printf("\tIntroduza o nome do ficheiro: ");
76          scanf("%s",fnome);
77
78          criaFile(atoi(linhas),atoi(colunas),fnome);
79          opcoes(d1);
80    }
81    if(strcmp(opcao,"2")==0)
82    {
83          char *nomef=(char *)calloc(100,sizeof(char));
84
85          printf("\tIntroduza o nome do ficheiro existente: ");
86          scanf("%s",nomef);
87
88          if(load(nomef,d1))
89          {
90                  puts("FICHEIRO EXISTE!");
91                  opcoes(d1);
92                  save("out.pbm",d1);
93          }
94          else printf("\n\n> ERRO, ficheiro nao existe!\n\n");
95    }
96    if(strcmp(opcao,"3")==0)
97          save("out.pbm",d1);
98    if(strcmp(opcao,"4")==0)
99    {
100         printf("\tIntroduza a marca: ");
101         scanf("%s",marca);
102         setMarca(atoi(marca));
103   }
104   if(strcmp(opcao,"0")==0)
105         exit(1);
106   else menu(d1);
107 }
108
109 /*
110   Permite inserir uma lista de comandos separados por '\n'
111   ate se carregar em 's'.
112       Comandos disponiveis:
113         - Point    x,y
114         - PolyLine x1,y1,...,xn,yn
115         - Line     x1,y1,x2,y2
116         - Rect     x1,y1,x2,y2
117         - Circ     x,y,r
118         - Elip     x1,y1,x2,y2
119 */
120
121 void opcoes (DESENHO d1)
122 {
123   int a,b,c,d;
124   char *comando="" , *pontos="";
125
126   while((comando[0]!='s') && (pontos[0]!='s'))
127   {
128         //system("clear");
129         printDesenho(d1);
130         puts("\tPrima 's' para voltar\n\n\n");
```

```
131
132         char *pontos2=(char *)calloc(100,sizeof(char));
133         comando=(char *)calloc(100,sizeof(char));
134         pontos=(char *)calloc(100,sizeof(char));
135
136         scanf("%s",comando);
137         scanf("%s",pontos);
138
139         strcpy(pontos2,pontos);
140
141         if(strcmp(comando,"Point")==0 && valida(pontos,2))
142         {
143                 sscanf (pontos,"%d,%d", &a , &b);
144                 point(a,b,d1);
145         }
146         if(strcmp(comando,"PolyLine")==0)
147         {
148                 float mtz[100];
149
150                 for(b=0 ; pontos2-1!=NULL ; b++ , pontos2 = strchr(pontos2,',')+1)
151                 {
152                         char *acc = (char *)calloc(100,sizeof(char));
153
154                         sscanf (pontos2,"%[^,]", acc);
155                         mtz[b] = atoi(acc);
156                 }
157                 polyLine(mtz,b-1,d1);
158                 comando=""; // comando = 'PolyLine' contem 'Line'
159         }
160         if(strcmp(comando,"Line")==0 && valida(pontos,4))
161         {
162                 sscanf (pontos,"%d,%d,%d,%d", &a , &b , &c , &d);
163                 line(a,b,c,d,d1);
164         }
165         if(strcmp(comando,"Rect")==0 && valida(pontos,4))
166         {
167                 sscanf (pontos,"%d,%d,%d,%d", &a , &b , &c , &d);
168                 rect(a,b,c,d,d1);
169         }
170         if(strcmp(comando,"Circ")==0 && valida(pontos,3))
171         {
172                 sscanf (pontos,"%d,%d,%d", &a , &b , &c);
173                 if(!(circ(a,b,c,d1))) puts("Comando invalido!");
174         }
175         if(strcmp(comando,"Elip")==0 && valida(pontos,4))
176         {
177                 sscanf (pontos,"%d,%d,%d,%d", &a , &b , &c , &d);
178                 if(!(elip(a,b,c,d,d1))) puts("Comando invalido!");
179         }
180   }
181 }
182
183 void criaFile(int l,int c,char fnome[])
184 {
185   FILE *fp=fopen(fnome,"w");
186   int i,j;
187
188   fprintf(fp,"P1\n");
189   fprintf(fp,"#Imagem PBM gerada pela aplicacao de DESENHO\n");
190   fprintf(fp,"%d %d\n",c,l);
191   for(i=0;i<l;i++)
192   {
193         for(j=0;j<c;j++)
194         {
195                 fprintf(fp,"0 ");
196         }
197         fprintf(fp,"\n");
198   }
```

```
199
200   fclose(fp);
201 }
202
203 /*
204    validacao dos parametros das funcoes
205 */
206
207 int valida(char *op, int n)
208 {
209   int i,ret=0;
210
211   for(i=0;op-1!=NULL;i++,op = strchr(op,',')+1);
212   if (i==n) ret=1;
213
214   return ret;
215 }
216
217 void setMarca(int n)
218 {
219         marca=n;
220 }
221
222 /* ------------------------------------------------------------
223    Inicio do codigo das funcoes de desenho
224    ------------------------------------------------------------ */
225
226 /*
227    x : linha da matriz d onde se quer marcar o ponto
228    y : coluna da matriz d onde se quer marcar o ponto
229    d : matriz contendo o desenho onde se vai marcar o ponto
230 */
231
232 void point(int x, int y, DESENHO d)
233 {
234         d[x][y]=marca;
235 }
236
237 /*
238    a : array contendo todos os pontos na forma:
239          x1,y1,x2,y2,...,xn,yn
240    b : tamanho-1 do array a
241    d : matriz contendo o desenho onde se vai marcar as varias linhas
242 */
243
244 void polyLine(float a[], int b, DESENHO d)
245 {
246   int i;
247
248   for(i=0;i<b-2;i+=2)
249         line(a[i],a[i+1],a[i+2],a[i+3],d);
250 }
251
252 /*
253    Uma imagem explicativa:
254
255    +-------------------------------------------+
256    |                                           |
257    | desenho d                      (x2,y2)    |
258    |                                  |        |
259    |                                +---+      |
260    |                              +---+        |
261    |                            +---+          |
262    |                          +---+            |
263    |                        +---+              |
264    |                    +---+                  |
265    |                +---+                      |
266    |    (x1,y1)-|                              |
```

```
267     |                                           |
268     +-------------------------------------------+
269   */
270
271
272   void line(float x1, float y1, float x2, float y2, DESENHO d)
273   {
274     float m = (y1-y2)/(x1-x2), b = y1 - m * x1, t, tx,ty;
275     int i, y, x;
276
277     if(x1!=x2)
278     {
279           if(x1>x2)
280           {
281                   ty=y1; tx=x1; x1 = x2; y1 = y2; x2 = tx; y2 = ty;
282           }
283
284           for(i=x1;i<x2;i++)
285           {
286                   y = m * i + b;
287                   d[i][y]=marca;
288           }
289     }
290     else if(y1 != y2)
291     {
292           if (y1 > y2)
293           {
294                   t = y2; y2 = y1; y1 = t; t = x2; x2 = x1; x1 = t;
295           }
296           m = (x1 - x2) / (y1 - y2);
297           b = x1 - m * y1;
298
299           for(i=y1;i<=y2;i++)
300           {
301                   x = (m * i + b);
302                   d[x][i]=marca;
303           }
304     }
305   }
306
307   /*
308      Desenha um rectangulo cuja sua diagonal seja
309      uma linha do ponto (x1,y1) ate (x2,y2)
310
311      Uma imagem explicativa:
312
313      +-------------------------------------------+
314      | desenho d                                 |
315      |                                (x2,y2)    |
316      |            +----------------------+       |
317      |            |                   l  |       |
318      |            |                a  +--+       |
319      |            |             o n +--+  |       |
320      |            |          g  +--+     |       |
321      |            |       a  +--+        |       |
322      |            |    i  +--+           |       |
323      |            |d  +--+               |       |
324      |            |+--+                  |       |
325      |            +----------------------+       |
326      |       (x1,y1)                             |
327      +-------------------------------------------+
328
329   */
330
331   void rect(int x1, int y1, int x2, int y2, DESENHO d)
332   {
333           line(x1,y1,x2,y1,d); // linha vertical esquerda
334           line(x1,y2,x2,y2,d); // linha vertical direita
```

```
335         line(x2,y1,x2,y2,d); // linha horizontal inferior
336         line(x1,y1,x1,y2,d); // linha horizontal superior
337 }
338
339
340 /*
341   Propriedades para desenhar um circulo:
342     a)  'l' e 'c'  SEMPRE >= que 'r'
343     b1) 'l'+'r'    SEMPRE <= MAXLINHAS
344     b2) 'c'+'r'+1  SEMPRE <= MAXCOLUNAS
345 */
346
347 int circ(int l , int c , int r , DESENHO d)
348 {
349   int a,ret=0,linhas=d[0][0],colunas=d[0][1];
350
351   if((l>=r && c>=r) || (l+r<=linhas && c+r+1<=colunas))
352   {
353         for(a=0;a<360*r;a++)
354         {
355                 int px = r * cos(a) , py = r * sin(a) , yy = py+c , xx = px+l;
356
357                 d[xx][yy]=marca;
358         }
359         ret=1;
360   }
361   return ret;
362 }
363
364 /*
365   Propriedades para desenhar uma elipse:
366     a1)  'x1' e 'y1' SEMPRE >= x2
367     b1) 'x1'+'x2'    SEMPRE <= MAXLINHAS
368     b2) 'y1'+'y2'+1  SEMPRE <= MAXCOLUNAS
369 */
370
371 int elip(int x1, int y1, int x2, int y2, DESENHO d)
372 {
373   int a,ret=0,linhas=d[0][0],colunas=d[0][1];
374
375   if((x1>=x2 && y1>=y2) || (x1+x2<=linhas && y1+y2+1<=colunas))
376   {
377         for(a=0;a<360*(x2+y2);a++)
378         {
379                 int px = x2 * cos(a) , py = y2 * sin(a) , yy = py+y1 , xx = px+x1;
380
381                 d[xx][yy]=marca;
382         }
383         ret=1;
384   }
385   return ret;
386 }
387
388 /* ------------------------------------------------------------
389    Fun
390    ------------------------------------------------------------ */
391
392 /* desenha uma cara numa imagem 800x600
393   circ(250,300,250,d1);
394   circ(300,300,50,d1);
395   elip(200,200,50,100,d1);
396   elip(200,400,50,100,d1);
397   elip(400,300,30,100,d1);
398   elip(400,300,1,100,d1);
399 */
```

B Código C fornecido pelo docente

B.1 Código das funções load, save e printDesenho

```
/* --------------------------------
   myio.c
   --------------------------------
*/

/* ------------------------------------------------
   Parte do codigo do TP2 de 2006
   2006-03-10: created by jcr
   ------------------------------------------------ */

#include <stdio.h>
#include "mydatatype.h"
#include "bool.h"

/*
   --- Funcao de leitura de um desenho previamente gravado em PBM (P1)

         O resultado da funcao e um booleano:
                1 : o desenho foi lido com sucesso
                0 : o desenho nao foi lido, ocorreu um erro
*/

boolean load( char fnome[], DESENHO d )
{

/*
   fnome: nome do ficheiro que se vai carregar
   d    : matriz para onde vai ser carregado o desenho
*/

  FILE *fp;
  char linha[MAXCOLUNAS+1];  // usada para ler as linhas de comentario
  int i, j;                  // variaveis de controlo das linhas e colunas
  boolean erro=FALSE;        // a primeira linha da matriz guardara a dim da imagem

  fp = fopen(fnome,"r");
  if(!fp) return FALSE;
  else
  {
        fgets(linha,MAXCOLUNAS+1,fp); /* le a linha com o identificador */

        /* Enquanto nao conseguir ler as dimensoes... */
        while(fscanf(fp,"%d %d",&(d[0][1]),&(d[0][0]))!=2)
          fgets(linha,MAXCOLUNAS+1,fp); // descarta as linhas com comentarios

        i=1;
        while((i<d[0][0])&&(!erro)) //Enq nao se tiverem lido as linhas todas
        {
          j=0;
          while((j<d[0][1])&&(!erro)) // Enq nao se tiverem lido todas as colunas
                if(!fscanf(fp,"%d",&(d[i][j]))) erro = TRUE;
                else j++;
          i++;
        }

        fclose(fp);
        return (!erro);
  }
}

/*
    --- Funcao de gravacao do desenho carregado na matriz para o formato PBM (P1)
```

```
63
64              O resultado da funcao e um booleano:
65                    1 : o desenho foi gravado com sucesso
66                    0 : o desenho nao foi gravado, ocorreu um erro
67  */
68
69  boolean save( char fnome[], DESENHO d )
70  {
71
72  /*
73     fnome: nome do ficheiro onde se vai gravar
74     d    : matriz contendo o desenho que se vai gravar
75  */
76
77    FILE *fp;
78    int linhas = d[0][0] , colunas = d[0][1] , i , j;
79     // variaveis de controlo das linhas e colunas
80     // a primeira linha da matriz contem a dim da imagem
81
82    fp = fopen(fnome,"w");
83    if(!fp) return FALSE;
84    else
85    {
86          fprintf(fp,"P1\n");
87          fprintf(fp,"# Imagem PBM gerada pela aplicacao de DESENHO\n");
88          fprintf(fp,"%d %d\n",d[0][1],d[0][0]);
89
90          for(i=1;i<=linhas;i++)
91          {
92                  for(j=1;j<=colunas;j++)
93                  {
94                          fprintf(fp,"%d ",d[i][j]);
95                  }
96                  fprintf(fp,"\n");
97          }
98
99          fclose(fp);
100         return (TRUE);
101   }
102 }
103
104 /* --- Funcao que escreve a matriz no monitor */
105
106 int printDesenho(DESENHO d)
107 {
108   int linhas = d[0][0] , colunas = d[0][1] , i , j;
109
110   for(i=1;i<=linhas;i++)
111   {
112         for(j=1;j<=colunas;j++)
113         {
114                 printf("%d",d[i][j]);
115         }
116         printf("\n");
117   }
118   return 0;
119 }
```

B.2 Ficheiro bool.h

```
1  #ifndef _BOOL
2  #define BOOL
3  #define boolean int
4  #define TRUE 1
5  #define FALSE 0
6  #endif
```

B.3 Ficheiro mydatatype.h

```
1  #ifndef _MYDATATYPE
2  #define _MYDATATYPE
3
4  /* --- Definicoes globais para a definicao da matriz */
5  #define MAXLINHAS 601
6  #define MAXCOLUNAS 800
7  /* --- Tipo de Dados para a Matriz */
8  typedef int DESENHO[MAXLINHAS][MAXCOLUNAS];
9
10 #endif
```