```
 1  #ifndef _CMDLINE_H_INCLUDED_
 2  #define _CMDLINE_H_INCLUDED_
 3
 4  #include <string>
 5  #include <iostream>
 6
 7  #define OPT_DEFAULT   0
 8  #define OPT_SEEN      1
 9  #define OPT_MANDATORY 2
10
11  struct option_t {
12      bool has_arg;
13      const char *short_name;
14      const char *long_name;
15      const char *def_value;
16      void (*parse)(std::string const &);
17      int flags;
18  };
19
20  class cmdline {
21      // Este atributo apunta a la tabla que describe todas
22      // las opciones a procesar. Por el momento, sólo puede
23      // ser modificado mediante constructor, y debe finalizar
24      // con un elemento nulo.
25      //
26      option_t *option_table;
27
28      // El constructor por defecto cmdline::cmdline(), es
29      // privado, para evitar construir parsers sin opciones.
30      //
31      cmdline();
32      int do_long_opt(const char *, const char *);
33      int do_short_opt(const char *, const char *);
34
35  public:
36      cmdline(option_t *);
37      void parse(int, char * const []);
38  };
39
40  #endif
```

```
 1  // cmdline − procesamiento de opciones en la línea de comando.
 2  //
 3  // $Date: 2012/09/14 13:08:33 $
 4  //
 5  #include <string>
 6  #include <cstdlib>
 7  #include <iostream>
 8  #include "cmdline.h"
 9
10  using namespace std;
11
12  cmdline::cmdline()
13  {
14  }
15
16  cmdline::cmdline(option_t *table) : option_table(table)
17  {
18  }
19
20  void
21  cmdline::parse(int argc, char * const argv[])
22  {
23  #define END_OF_OPTIONS(p)   \
24      ((p)->short_name == 0   \
25       && (p)->long_name == 0 \
26       && (p)->parse == 0)
27
28      // Primer pasada por la secuencia de opciones: marcamos
29      // todas las opciones, como no procesadas. Ver código de
30      // abajo.
31      //
32      for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
33          op->flags &= ~OPT_SEEN;
34
35      // Recorremos el arreglo argv. En cada paso, vemos
36      // si se trata de una opción corta, o larga. Luego,
37      // llamamos a la función de parseo correspondiente.
38      //
39      for (int i = 1; i < argc; ++i) {
40          // Todos los parámetros de este programa deben
41          // pasarse en forma de opciones. Encontrar un
42          // parámetro no−opción es un error.
43          //
44          if (argv[i][0] != '−') {
45              cerr << "Invalid non−option argument: "
46                   << argv[i]
47                   << endl;
48              exit(1);
49          }
50
51          // Usamos "−−" para marcar el fin de las
52          // opciones; todo los argumentos que puedan
53          // estar a continuación no son interpretados
54          // como opciones.
55          //
56          if (argv[i][1] == '−'
57              && argv[i][2] == 0)
58              break;
59
60          // Finalmente, vemos si se trata o no de una
61          // opción larga; y llamamos al método que se
62          // encarga de cada caso.
63          //
64          if (argv[i][1] == '−')
65              i += do_long_opt(&argv[i][2], argv[i + 1]);
66          else
67              i += do_short_opt(&argv[i][1], argv[i + 1]);
68      }
69
70      // Segunda pasada: procesamos aquellas opciones que,
71      // (1) no hayan figurado explícitamente en la línea
72      // de comandos, y (2) tengan valor por defecto.
73      //
```

```
74      for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
75  #define OPTION_NAME(op) \
76      (op->short_name ? op->short_name : op->long_name)
77          if (op->flags & OPT_SEEN)
78              continue;
79          if (op->flags & OPT_MANDATORY) {
80              cerr << "Option "
81                   << "-"
82                   << OPTION_NAME(op)
83                   << " is mandatory."
84                   << "\n";
85              exit(1);
86          }
87          if (op->def_value == 0)
88              continue;
89          op->parse(string(op->def_value));
90      }
91  }
92
93  int
94  cmdline::do_long_opt(const char *opt, const char *arg)
95  {
96      option_t *op;
97
98      // Recorremos la tabla de opciones, y buscamos la
99      // entrada larga que se corresponda con la opción de
100     // línea de comandos. De no encontrarse, indicamos el
101     // error.
102     //
103     for (op = option_table; op->long_name != 0; ++op) {
104         if (string(opt) == string(op->long_name)) {
105             // Marcamos esta opción como usada en
106             // forma explícita, para evitar tener
107             // que inicializarla con el valor por
108             // defecto.
109             //
110             op->flags |= OPT_SEEN;
111
112             if (op->has_arg) {
113                 // Como se trada de una opción
114                 // con argumento, verificamos que
115                 // el mismo haya sido provisto.
116                 //
117                 if (arg == 0) {
118                     cerr << "Option requires argument: "
119                          << "--"
120                          << opt
121                          << "\n";
122                     exit(1);
123                 }
124                 op->parse(string(arg));
125                 return 1;
126             } else {
127                 // Opción sin argumento.
128                 //
129                 op->parse(string(""));
130                 return 0;
131             }
132         }
133     }
134
135     // Error: opción no reconocida. Imprimimos un mensaje
136     // de error, y finalizamos la ejecución del programa.
137     //
138     cerr << "Unknown option: "
139          << "--"
140          << opt
141          << "."
142          << endl;
143     exit(1);
144
145     // Algunos compiladores se quejan con funciones que
146     // lógicamente no pueden terminar, y que no devuelven
```

```
147      // un valor en esta última parte.
148      //
149      return -1;
150  }
151
152  int
153  cmdline::do_short_opt(const char *opt, const char *arg)
154  {
155      option_t *op;
156
157      // Recorremos la tabla de opciones, y buscamos la
158      // entrada corta que se corresponda con la opción de
159      // línea de comandos. De no encontrarse, indicamos el
160      // error.
161      //
162      for (op = option_table; op->short_name != 0; ++op) {
163          if (string(opt) == string(op->short_name)) {
164              // Marcamos esta opción como usada en
165              // forma explícita, para evitar tener
166              // que inicializarla con el valor por
167              // defecto.
168              //
169              op->flags |= OPT_SEEN;
170
171              if (op->has_arg) {
172                  // Como se trata de una opción
173                  // con argumento, verificamos que
174                  // el mismo haya sido provisto.
175                  //
176                  if (arg == 0) {
177                      cerr << "Option requires argument: "
178                           << "-"
179                           << opt
180                           << "\n";
181                      exit(1);
182                  }
183                  op->parse(string(arg));
184                  return 1;
185              } else {
186                  // Opción sin argumento.
187                  //
188                  op->parse(string(""));
189                  return 0;
190              }
191          }
192      }
193
194      // Error: opción no reconocida. Imprimimos un mensaje
195      // de error, y finalizamos la ejecución del programa.
196      //
197      cerr << "Unknown option: "
198           << "-"
199           << opt
200           << "."
201           << endl;
202      exit(1);
203
204      // Algunos compiladores se quejan con funciones que
205      // lógicamente no pueden terminar, y que no devuelven
206      // un valor en esta última parte.
207      //
208      return -1;
209  }
```

```
 1  #ifndef COMPLEJO_H
 2  #define COMPLEJO_H
 3
 4  #include<iostream>
 5  using namespace std;
 6
 7  class complejo
 8
 9  {
10  private:
11
12      double x,y;
13
14  public:
15
16      complejo();
17
18      complejo(double , double );
19
20      complejo(const complejo &);
21
22      ~complejo();
23
24      double GetReal()const;
25
26      double GetImag()const;
27
28      void SetReal(double xx);
29
30      void SetImag(double yy);
31
32      friend ostream& operator<<(ostream&, const complejo &);
33
34      friend istream& operator>>(istream&,complejo &);
35
36      complejo& operator=(const complejo &);
37
38      const complejo operator+(const complejo &);
39
40      const complejo operator+(const double);
41
42      void identidadDesde(const complejo &); //f(z) = z
43
44      void exponencialDesde(const complejo &); //f(z) = e^z
45
46      void cuadradoDesde(const complejo &); //f(z) = z^2
47
48      void cuboDesde(const complejo &); //f(z) = z^3
49
50      void senoDesde(const complejo &); //f(z) = sin(z)
51  };
52
53  #endif
```

```cpp
#include<iostream>
#include<math.h>
#include "complejo.h"

using namespace std;

complejo::complejo () : x(0.0) , y(0.0) {}

complejo::complejo (const complejo & c) : x(c.x) , y (c.y) {}

complejo::complejo (double a, double b): x(a) , y(b) {}

complejo::~complejo() {}

ostream& operator<<(ostream &os, const complejo &c){
    os<<"("<<c.x<<","<<c.y<<")"<<endl;
    return os;
}

istream& operator>>(istream &is,complejo &c){
    bool good=false;
    double r=0,i=0;
    char ch=0;

    if(is>>ch && ch=='('){
        if(is>>r && is>>ch && ch==',' && is>>i && is>>ch && ch==')'){
            good=true;
        } else{
            good=false;
        }
    } else if(is.good()){
        is.putback(ch);
        if(is>>r)
            good=true;
        else
            good=false;
    }

    if(good){
        c.x=r;
        c.y=i;
    } else{
        is.clear(ios::badbit);
    }

    return is;
}

complejo& complejo::operator=(const complejo & b){
    x = b.x;
    y= b.y;
    return *this;
}

double complejo::GetReal()const {
    return x;
}

double complejo::GetImag()const {
    return y;
}

void complejo::SetReal(double xx){
    x=xx;
}

void complejo::SetImag(double yy){
    y=yy;
}

const complejo complejo::operator+(const complejo &r){
    return complejo (x+r.x , y+r.y);
}
```

```
 74
 75 const complejo complejo::operator+(const double f){
 76     return complejo (x+f,y);
 77 }
 78
 79 //z
 80 void complejo::identidadDesde(const complejo &c){
 81     x = c.x;
 82     y = c.y;
 83 }
 84
 85 //e^z = e^(x+iy) = e^x * e^(iy) = e^x * (cos(y) + i sin(y))
 86 void complejo::exponencialDesde(const complejo &c){
 87     x = exp(c.x)*cos(c.y);
 88     y = exp(c.x)*sin(c.y);
 89 }
 90
 91 // z^2 = (x+iy)^2 = x^2 - y^2 + i 2*x*y
 92 void complejo::cuadradoDesde(const complejo &c){
 93     x = c.x*c.x - c.y*c.y;
 94     y = 2*c.x*c.y;
 95 }
 96
 97 // z^3 = (x+iy)^3 = x^3 - 3*x*y^2 - i (y^3 - 3*x^2*y)
 98 void complejo::cuboDesde(const complejo &c){
 99     x = c.x*c.x*c.x - 3*c.x*c.y*c.y;
100     y = 3*c.x*c.x*c.y - c.y*c.y*c.y;
101 }
102
103 // sin(z) = sin(x+iy) = sin(x)*cosh(y) + i cos(x)sinh(y)
104 void complejo::senoDesde(const complejo &c){
105     x = sin(c.x)*cosh(c.y);
106     y = cos(c.x)*sinh(c.y);
107 }
```

```cpp
#ifndef PGMIMAGE_H
#define PGMIMAGE_H

#include <iostream>
#include <limits>
using namespace std;

// Limitación en la profundidad del color para ahorrar memoria usando uchar
#define MIN_COLOR_DEPTH   1
#define MAX_COLOR_DEPTH 255
typedef unsigned char pixel_t;



/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||||| PGMimage ||||||||||||||||||||||||||||||||||||||||||*/
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
class PGMimage
{
    static const string MagicNumber; // Número mágico PGM
    size_t              Width;       // Ancho del lienzo (imagen)
    size_t              Height;      // Alto del lienzo (imagen)
    pixel_t             ColorDepth;  // Profundidad del color (niveles)
    pixel_t             **canvas;    // Lienzo, matriz de imagen


    ////////////////////////// Utilidades internas ////////////////////////////

    // Ignorar comentarios estilo PGM en un stream
    static void ignore_comments(istream &);

    // Pedir memoria para un lienzo de w x h
    static pixel_t** new_canvas(size_t, size_t);

    // Limitar profundidad de color
    static void validate_color_depth(pixel_t &);

    // Destruir el lienzo sobre el objeto actual
    void canvasDestroy();


public:
    // 1) Constructor (con sus argumentos por defecto)
    PGMimage(size_t w=1, size_t h=1, pixel_t d=MAX_COLOR_DEPTH);

    // 2) Constructor por copia
    PGMimage(const PGMimage &);

    // 3) Destructor
    ~ PGMimage() { canvasDestroy(); };

    // 4) Indexación del lienzo (l-value y r-value: c[y][x])
    pixel_t* operator[](size_t) const;

    // 5-7) Obtención de ancho, alto, profundidad de color
    size_t  getWidth() const;
    size_t  getHeight() const;
    pixel_t getColorDepth() const;

    // 8) Cambio de profundidad de color
    void setColorDepth(pixel_t);

    // 9) Cambio de tamaño de imagen
    void resize(size_t, size_t);

    // 10) Impresión en flujo/archivo/stdin
    friend ostream & operator<<(ostream &, const PGMimage &);

    // 11) Carga desde flujo/archivo/stdin
    friend istream & operator>>(istream &, PGMimage &);
};


```

```
74  #endif // PGMIMAGE_H
```

```cpp
#include "PGMimage.h"
const string PGMimage::MagicNumber = "P2";



/*|////////////////////////////////|   1)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|//////////////////////////////| Constructor |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
PGMimage::PGMimage(size_t w, size_t h, pixel_t d)
{
    this->Width  = w;
    this->Height = h;

    // Limitación en la profundidad de color
    PGMimage::validate_color_depth(d);
    this->ColorDepth = d;

    // Memoria para el lienzo
    this->canvas = PGMimage::new_canvas(this->Width, this->Height);

    // Inicialización en 0
    for (size_t i = 0; i < h; i++)
        for (size_t j = 0; j < w; j++)
            this->canvas[i][j] = 0;
}



/*|////////////////////////////////|   2)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////| Constructor por copia |\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
PGMimage::PGMimage(const PGMimage &o)
{
    this->Width  = o.Width;
    this->Height = o.Height;
    this->ColorDepth = o.ColorDepth;

    // Memoria para la copia
    this->canvas = PGMimage::new_canvas(this->Width, this->Height);

    // Copia de los datos
    for (size_t i = 0; i < this->Height; i++)
        for (size_t j = 0; j < this->Width; j++)
            this->canvas[i][j] = o.canvas[i][j];
}



/*|////////////////////////////////|   4)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|//////////| Indexación del lienzo (l-value y r-value: c[y][x]) |\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
pixel_t* PGMimage::operator[](size_t y) const
{
    if (y >= this->Height)
        return this->canvas[this->Height-1];

    return this->canvas[y];
}



/*|////////////////////////////////|  5-7) |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////| Obtención de ancho, alto, profundidad de color |\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
size_t  PGMimage::getWidth() const      { return this->Width;      }
size_t  PGMimage::getHeight() const     { return this->Height;     }
pixel_t PGMimage::getColorDepth() const { return this->ColorDepth; }



/*|////////////////////////////////|   8)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////| Cambio de profundidad de color |\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
```

```
 74  void PGMimage::setColorDepth(pixel_t d)
 75  {
 76      PGMimage::validate_color_depth(d);
 77
 78      // Cálculo de factor de amplificación y actualización de la profundidad
 79      float amp = (float) d / (float) this->ColorDepth;
 80      this->ColorDepth = d;
 81
 82      // Cálculo de los datos con la nueva profundidad
 83      for (size_t i = 0; i < this->Height; i++)
 84          for (size_t j = 0; j < this->Width; j++)
 85              this->canvas[i][j] *= amp;
 86  }
 87
 88
 89
 90  /*|//////////////////////////////|   9)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
 91  /*|//////////////////////| Cambio de tamaño de imagen |\\\\\\\\\\\\\\\\\\\\\\|*/
 92  /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
 93  void PGMimage::resize(size_t w, size_t h)
 94  {
 95      size_t w_max, h_max;
 96      pixel_t **auxcnv;
 97
 98      // Memoria para el nuevo lienzo
 99      auxcnv = PGMimage::new_canvas(w, h);
100
101      // Copiado de los datos, con posible pérdida por recorte
102      w_max = min(w, this->Width);
103      h_max = min(h, this->Height);
104
105      for (size_t i = 0; i < h_max; i++)
106          for (size_t j = 0; j < w_max; j++)
107              auxcnv[i][j] = this->canvas[i][j];
108
109      // Liberación de la memoria antigua
110      canvasDestroy();
111
112      // Actualización de los valores
113      this->Width = w;
114      this->Height = h;
115      this->canvas = auxcnv;
116  }
117  // NOTA: no se realiza un escalado de la imagen, solo se modifica el lienzo,
118  // resultando en posibles recortes o agregado de pixels con eventual basura.
119
120
121
122  /*|//////////////////////////////|  10)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
123  /*|//////////////////| Impresión en flujo/archivo/stdin |\\\\\\\\\\\\\\\\\\|*/
124  /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
125  ostream & operator<<(ostream &os, const PGMimage &c)
126  {
127      // Encabezado del archivo
128      os << c.MagicNumber << endl;
129      os << c.Width << ' ' << c.Height << endl;
130      os << (size_t) c.ColorDepth << endl;
131
132      // Datos de pixels
133      for (size_t i = 0; i < c.Height; i++)
134      {
135          os << (size_t) c.canvas[i][0];
136          for (size_t j = 1; j < c.Width; j++)
137              os << ' ' << (size_t) c.canvas[i][j];
138
139          os << endl;
140      }
141
142      return os;
143  }
144
145
146
```

```
147 /*|////////////////////////////////|  11)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
148 /*|//////////////////| Carga desde flujo/archivo/stdin |\\\\\\\\\\\\\\\\\\\\|*/
149 /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
150 istream & operator>>(istream &is, PGMimage &c)
151 {
152     size_t w = 0, h = 0, aux = 0, i, j;
153     pixel_t d = 0, **auxcnv;
154     bool errors = true;
155 
156     // Lectura del encabezado
157     int mnL = c.MagicNumber.length() + 1;
158     char *mn = new char[mnL];
159     is.get(mn, mnL);
160     // Número mágico
161     if ( mn == c.MagicNumber )
162     {
163         PGMimage::ignore_comments(is);
164         // Ancho y alto
165         if (is >> w && is >> h)
166         {
167             PGMimage::ignore_comments(is);
168             // Profundidad de color
169             if (is >> aux)
170             {
171                 d = aux;
172                 errors = false;
173             }
174         }
175     }
176     delete mn;
177 
178     // Lectura de los datos de pixel
179     if (!errors)
180     {
181         // Limitar profundidad de color
182         PGMimage::validate_color_depth(d);
183 
184         // Memoria para el lienzo
185         auxcnv = PGMimage::new_canvas(w, h);
186 
187         // Carga de datos
188         for (i = 0; i < h; i++)
189         {
190             for (j = 0; j < w; j++)
191             {
192                 PGMimage::ignore_comments(is);
193                 if (is >> aux) auxcnv[i][j] = aux;
194                 else { errors = true; break; }
195             }
196             if (errors) break;
197         }
198 
199         // Si no ha fallado la carga de valores
200         if (!errors)
201         {
202             // Actualización del objeto
203             c.canvasDestroy();
204             c.canvas = auxcnv;
205             c.Width = w;
206             c.Height = h;
207             c.ColorDepth = d;
208         }
209         // En caso de falla, se deja el objeto intacto y se destruye auxcnv
210         else
211         {
212             for (i = 0; i < h; i++)
213                 delete [] auxcnv[i];
214             delete [] auxcnv;
215         }
216     }
217 
218     if (errors) // Si hubo errores, se indica en el stream
219         is.clear(ios::badbit);
```

```
220
221     return is;
222 }
223
224
225
226 /*|//////////////////////////////|   *)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
227 /*|//////////////////////////| Utilidades internas |\\\\\\\\\\\\\\\\\\\\\\\\|*/
228 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
229
230 // Ignorar comentarios estilo PGM en un stream
231 void PGMimage::ignore_comments(istream &s)
232 {
233     char ch;
234     while (s >> ch)
235     {
236         if (ch == '#')
237         {
238             s.ignore(numeric_limits<streamsize>::max(), '\n');
239         }
240         else
241         {
242             s.putback(ch);
243             break;
244         }
245     }
246 }
247
248 // Pedir memoria para un lienzo de w x h
249 pixel_t** PGMimage::new_canvas(size_t w, size_t h)
250 {
251     pixel_t **cnv = new pixel_t* [h];
252
253     for (size_t i = 0; i < h; i++)
254         cnv[i] = new pixel_t[w];
255
256     return cnv;
257 }
258
259 // Limitar profundidad de color
260 void PGMimage::validate_color_depth(pixel_t &d)
261 {
262     if (d > MAX_COLOR_DEPTH) d = MAX_COLOR_DEPTH;
263     if (d < MIN_COLOR_DEPTH) d = MIN_COLOR_DEPTH;
264 }
265
266 // Destruir el lienzo sobre el objeto actual
267 void PGMimage::canvasDestroy()
268 {
269     for (size_t i = 0; i < this->Height; i++)
270         delete [] this->canvas[i];
271
272     delete [] this->canvas;
273 }
```

```
 1  /*
 2   * File:   utils.h
 3   * Author: juanzaragoza
 4   *
 5   * Created on 13 de septiembre de 2014, 14:54
 6   */
 7  #ifndef UTILS_H
 8  #define UTILS_H
 9
10  #include <iostream>
11  #include <fstream>
12  #include <cstdlib>
13  #include "complejo.h"
14  #include "cmdline.h"
15  using namespace std;
16
17
18  // Porción del plano complejo a utilizar
19  #define MAP_X 2  // Ancho del mapeo centrado en el origen
20  #define MAP_Y 2  // Alto del mapeo centrado en el origen
21
22
23  // Tipo para los punteros a función compleja
24  typedef void (complejo::*function_t)(const complejo&);
25
26
27  ////////////////////// Variables globales de main.cpp ////////////////////////
28  extern option_t options[];            // Opciones CLA
29  extern function_t complex_function;   // Puntero a función compleja
30  extern istream *iss;                  // Puntero a stream de entrada
31  extern ostream *oss;                  // Puntero a stream de salida
32  extern fstream ifs;                   // Archivo de entrada
33  extern fstream ofs;                   // Archivo de salida
34  extern char *prog_name;               // Nombre del programa
35
36
37  ////////////////// Configuraciones de la función a aplicar ///////////////////
38  static const string functions_opts[] = // Argumentos de la opción -f
39  {
40      "z",
41      "exp(z)",
42      "z^2",
43      "z^3",
44      "sin(z)",
45      "" // No olvidar centinela de cadena vacía
46  };
47
48  static const function_t functions_ptrs[] = // Punteros a funciones complejas
49  {
50      &complejo::identidadDesde,
51      &complejo::exponencialDesde,
52      &complejo::cuadradoDesde,
53      &complejo::cuboDesde,
54      &complejo::senoDesde
55  };
56
57
58
59  /*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
60  /*|||||||||||||||||||||||||||||||| Utilidades ||||||||||||||||||||||||||||||||*/
61  /*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
62
63  // 1) CLA: Archivo de entrada
64  void opt_input(string const &);
65
66  // 2) CLA: Archivo de salida
67  void opt_output(string const &);
68
69  // 3) CLA: Función a aplicar
70  void opt_function(string const &);
71
72  // 4) CLA: Ayuda
73  void opt_help(string const &);
```

```
74
75  // 5) Obtener complejo asociado a los índices
76  void getComplexFromIndex(complejo &, size_t, size_t, size_t, size_t);
77
78  // 6) Obtener la fila asociada al complejo ( [i][ ] )
79  size_t getRowFromComplex(const complejo &, size_t);
80
81  // 7) Obtener la columna asociada al complejo ( [ ][j] )
82  size_t getColFromComplex(const complejo &, size_t);
83
84  #endif    /* UTILS_H */
```

```
 1  /*
 2   * File:   utils.cpp
 3   * Author: juanzaragoza
 4   *
 5   * Created on 13 de septiembre de 2014, 14:54
 6   */
 7  #include "utils.h"
 8
 9
10
11  /*|//////////////////////////////|   1)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
12  /*|////////////////////////| CLA: Archivo de entrada |\\\\\\\\\\\\\\\\\\\\\\\|*/
13  /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
14  void opt_input(string const &arg)
15  {
16      // Por defecto stdin, o bien archivo
17      if (arg == "-")
18      {
19          iss = &cin;
20      }
21      else
22      {
23          ifs.open(arg.c_str(), ios::in);
24          iss = &ifs;
25      }
26
27      // Comprobación de errores
28      if ( !iss->good() )
29      {
30          cerr << "Cannot open "
31               << arg
32               << "."
33               << endl;
34          exit(1);
35      }
36  }
37
38
39
40  /*|//////////////////////////////|   2)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
41  /*|////////////////////////| CLA: Archivo de salida |\\\\\\\\\\\\\\\\\\\\\\\|*/
42  /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
43  void opt_output(string const &arg)
44  {
45      // Por defecto stdout, o bien archivo
46      if (arg == "-")
47      {
48          oss = &cout;
49      }
50      else
51      {
52          ofs.open(arg.c_str(), ios::out);
53          oss = &ofs;
54      }
55
56      // Comprobación de errores
57      if ( !oss->good() )
58      {
59          cerr << "Cannot open "
60               << arg
61               << "."
62               << endl;
63          exit(1);
64      }
65  }
66
67
68
69  /*|//////////////////////////////|   3)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
70  /*|////////////////////////| CLA: Función a aplicar |\\\\\\\\\\\\\\\\\\\\\\\|*/
71  /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
72  void opt_function(string const &arg)
73  {
```

```
74      for (size_t i=0; functions_opts[i]!=""; i++)
75      {
76          // Si se encuentra la función en la lista de las disponibles
77          if(arg == functions_opts[i])
78          {
79              complex_function = functions_ptrs[i];
80              return;
81          }
82      }
83
84      // Si no se retornó antes, el parámetro es inválido
85      cerr << "Invalid function parameter: "
86           << arg
87           << "."
88           << endl;
89      exit(1);
90  }
91
92
93
94  /*|//////////////////////////////|   4)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
95  /*|//////////////////////////////| CLA: Ayuda |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
96  /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
97  void opt_help(string const &arg)
98  {
99      cout << "Usage: "
100          << prog_name
101          << " [-i file] [-o file] [-f function]"
102          << endl
103          << "Function list: "
104          << functions_opts[0];
105
106     for (size_t i=1; functions_opts[i]!=""; i++)
107         cout << ", "
108              << functions_opts[i];
109
110     cout << "."
111          << endl;
112     exit(0);
113 }
114
115
116
117 /*|//////////////////////////////|   5)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
118 /*|///////////////| Obtener complejo asociado a los índices |\\\\\\\\\\\\\\\|*/
119 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
120 void getComplexFromIndex(complejo &z, size_t i, size_t j,
121                          size_t h, size_t w)
122 {
123     if ( h && w && i < h && j < w)
124     {
125         z.SetReal( MAP_X * ( ((double)j + 0.5) / (double)w  -  0.5 ) );
126         z.SetImag( MAP_Y * ( 0.5  -  ((double)i + 0.5) / (double)h ) );
127     }
128 }
129
130
131
132 /*|//////////////////////////////|   6)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
133 /*|///////////| Obtener la fila asociada al complejo ( [i][ ] ) |\\\\\\\\\\\|*/
134 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
135 size_t getRowFromComplex(const complejo &z, size_t h)
136 {
137     return h * ( 0.5 - z.GetImag() / MAP_Y );
138 }
139
140
141
142 /*|//////////////////////////////|   7)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
143 /*|/////////| Obtener la columna asociada al complejo ( [ ][j] ) \\\\\\\\\\|*/
144 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
145 size_t getColFromComplex(const complejo &z, size_t w)
146 {
```

```
147       return w * ( 0.5 + z.GetReal() / MAP_X );
148   }
```

```cpp
#include <iostream>
#include <fstream>
#include "cmdline.h"
#include "PGMimage.h"
#include "complejo.h"
#include "utils.h"


////////////////////////////// Variables globales ////////////////////////////
option_t options[] =
{
    {true,  "i", "input",    "-",  opt_input,    OPT_DEFAULT},
    {true,  "o", "output",   "-",  opt_output,   OPT_DEFAULT},
    {true,  "f", "function", "z",  opt_function, OPT_DEFAULT},
    {false, "h", "help",     NULL, opt_help,     OPT_DEFAULT},
    {0, },
};
function_t complex_function = NULL;
istream *iss = NULL;
ostream *oss = NULL;
fstream ifs;
fstream ofs;
char *prog_name;



/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||| Main ||||||||||||||||||||||||||||||||||||*/
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
int main(int argc, char** argv)
{
    prog_name = argv[0];
    // Validación de argumentos
    cmdline cmdl(options);
    cmdl.parse(argc, argv);

    // Lectura del archivo de entrada
    PGMimage in_image;
    if ( !(*iss >> in_image) )
    {
        cerr << "Invalid PGM formated input." << endl;
        exit(1);
    }
    if (ifs) ifs.close();

    // Creación de una nueva imagen con las mismas dimensiones
    size_t h = in_image.getHeight();
    size_t w = in_image.getWidth();

    PGMimage out_image(w, h, in_image.getColorDepth());

    // Recorrido de la imagen y transformación
    complejo in_plane, out_plane;
    size_t i, j, row, col;

    for (i = 0; i < h; i++)
    {
        for (j = 0; j < w; j++)
        {
            // Pixel en la imagen de salida <-> Punto en el plano de salida
            getComplexFromIndex(out_plane, i, j, h, w);

            // Aplicación de la función, previamente seleccionada y apuntada
            (in_plane.*complex_function)(out_plane);

            // Punto en el plano de entrada <-> Pixel en la imagen de entrada
            row = getRowFromComplex(in_plane, h);
            col = getColFromComplex(in_plane, w);

            // Si no se cayó fuera de la imagen, se copia
            if (row < h && col < w)
            {
                out_image[i][j] = in_image[row][col];
```

```
74                  }
75              }
76          }
77
78          // Volcado en el archivo de salida
79          *oss << out_image;
80          if (ofs) ofs.close();
81
82          return 0;
83      }
```

```
# ////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ #
# |||||||||||||||||||||||||| Configuraciones |||||||||||||||||||||||||||||||||||||||| #
# \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\////////////////////////////////////// #

# Compilador:
CC = g++
# Flags para linkeo:
LFLAGS = -pedantic -Wall
# Flags para compilación:
CFLAGS = -ansi -pedantic-errors -Wall -O3
# Nombre de salida del proyecto:
OUT = tp0
# Directorio de archivos fuente:
SRC_DIR = src
# Directorio de archivos binarios:
BIN_DIR = bin
# Directorio de instalación:
INSTALL_DIR = /usr/bin



# ////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ #
# |||||||||||||||||||||||| Objetivos y dependencias |||||||||||||||||||||||||||||||||| #
# \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\////////////////////////////////////// #

    # ---------------------------- Códigos objeto ---------------------------- #
OBJECTS = $(BIN_DIR)/PGMimage.o  \
          $(BIN_DIR)/complejo.o  \
          $(BIN_DIR)/utils.o     \
          $(BIN_DIR)/cmdline.o   \
          $(BIN_DIR)/main.o


    # ---------------------- Reglas de construcción ---------------------- #
$(BIN_DIR)/$(OUT): $(OBJECTS) | $(BIN_DIR)
    $(CC) $(LFLAGS) $(OBJECTS) -o $(BIN_DIR)/$(OUT)

$(BIN_DIR)/PGMimage.o: $(SRC_DIR)/PGMimage.cpp \
                       $(SRC_DIR)/PGMimage.h   \
                     | $(BIN_DIR)

$(BIN_DIR)/complejo.o: $(SRC_DIR)/complejo.cpp \
                       $(SRC_DIR)/complejo.h   \
                     | $(BIN_DIR)

$(BIN_DIR)/utils.o:    $(SRC_DIR)/utils.cpp    \
                       $(SRC_DIR)/utils.h      \
                     | $(BIN_DIR)

$(BIN_DIR)/cmdline.o:  $(SRC_DIR)/cmdline.cpp  \
                       $(SRC_DIR)/cmdline.h    \
                     | $(BIN_DIR)

$(BIN_DIR)/main.o:     $(SRC_DIR)/main.cpp     \
                       $(SRC_DIR)/PGMimage.h   \
                       $(SRC_DIR)/complejo.h   \
                       $(SRC_DIR)/utils.h      \
                       $(SRC_DIR)/cmdline.h    \
                     | $(BIN_DIR)

$(BIN_DIR)/%.o: $(SRC_DIR)/%.cpp
    $(CC) -c $(CFLAGS) $< -o $@

$(BIN_DIR):
    mkdir $(BIN_DIR)



# ////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ #
# |||||||||||||||||||||||||| Utilidades extras |||||||||||||||||||||||||||||||||||||||| #
# \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\////////////////////////////////////// #
.PHONY: clean objclean deltemps install all remove purge

```

```
74      # ------------------------- Limpiar (todo) -------------------------- #
75  clean:
76      rm -rf $(BIN_DIR)
77
78      # ---------------------- Limpiar códigos objeto ---------------------- #
79  objclean:
80      rm -f $(BIN_DIR)/*.o
81
82      # ------------- Construir y eliminar archivos temporales ------------- #
83  deltemps: $(BIN_DIR)/$(OUT) objclean
84
85      # ------------------------------ Instalar ---------------------------- #
86  install: $(BIN_DIR)/$(OUT)
87      @ cp $(BIN_DIR)/$(OUT) "$(INSTALL_DIR)"
88      @ echo "'$(OUT)' --> Installed in: $(INSTALL_DIR)"
89
90      # --------------------- Todo (instalar y limpiar) ------------------- #
91  all: install clean
92
93      # ---------------------------- Desinstalar --------------------------- #
94  remove:
95      rm -f "$(INSTALL_DIR)/$(OUT)"
96
97      # ------------------ Purgar (desinstalar y limpiar) ------------------ #
98  purge: remove clean
```