```
 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      //
```

```
    for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
#define OPTION_NAME(op) \
    (op->short_name ? op->short_name : op->long_name)
        if (op->flags & OPT_SEEN)
            continue;
        if (op->flags & OPT_MANDATORY) {
            cerr << "Option "
                 << "-"
                 << OPTION_NAME(op)
                 << " is mandatory."
                 << "\n";
            exit(1);
        }
        if (op->def_value == 0)
            continue;
        op->parse(string(op->def_value));
    }
}

int
cmdline::do_long_opt(const char *opt, const char *arg)
{
    option_t *op;

    // Recorremos la tabla de opciones, y buscamos la
    // entrada larga que se corresponda con la opción de
    // línea de comandos. De no encontrarse, indicamos el
    // error.
    //
    for (op = option_table; op->long_name != 0; ++op) {
        if (string(opt) == string(op->long_name)) {
            // Marcamos esta opción como usada en
            // forma explícita, para evitar tener
            // que inicializarla con el valor por
            // defecto.
            //
            op->flags |= OPT_SEEN;

            if (op->has_arg) {
                // Como se trada de una opción
                // con argumento, verificamos que
                // el mismo haya sido provisto.
                //
                if (arg == 0) {
                    cerr << "Option requires argument: "
                         << "--"
                         << opt
                         << "\n";
                    exit(1);
                }
                op->parse(string(arg));
                return 1;
            } else {
                // Opción sin argumento.
                //
                op->parse(string(""));
                return 0;
            }
        }
    }

    // Error: opción no reconocida. Imprimimos un mensaje
    // de error, y finalizamos la ejecución del programa.
    //
    cerr << "Unknown option: "
         << "--"
         << opt
         << "."
         << endl;
    exit(1);

    // Algunos compiladores se quejan con funciones que
    // 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
 8  // Macro para elevar al cuadrado multiplicando por si mismo
 9  #define square(x) ((x)*(x))
10
11  class complex
12
13  {
14  private:
15
16      double _x, _y;
17
18  public:
19
20      complex();
21      complex(double , double);
22      complex(const complex &);
23
24      ~complex();
25
26      double getReal()const;
27      double getImag()const;
28
29      void setReal(double xx);
30      void setImag(double yy);
31
32      double getArg()const;
33      double getPha()const;
34
35      friend ostream& operator<<(ostream&, const complex &);
36      friend istream& operator>>(istream&, complex &);
37
38      complex& operator=(const complex &);
39      const complex operator+(const double)const;
40      void operator+=(const complex &);
41      void operator-=(const complex &);
42
43
44      // Operadores, operaciones binarias
45      static const complex  operator_add(const complex &, const complex &);
46      static const complex operator_subt(const complex &, const complex &);
47      static const complex operator_mult(const complex &, const complex &);
48      static const complex  operator_div(const complex &, const complex &);
49      static const complex  operator_pow(const complex &, const complex &);
50
51      // Funciones, operaciones unarias
52      static const complex exp(const complex &);
53      static const complex log(const complex &);
54      static const complex sin(const complex &);
55      static const complex cos(const complex &);
56
57      static const complex  real_as_complex(const complex &);
58      static const complex  imag_as_complex(const complex &);
59      static const complex   abs_as_complex(const complex &);
60      static const complex phase_as_complex(const complex &);
61  };
62
63
64  #endif
```

```cpp
#include<iostream>
#define _USE_MATH_DEFINES // Constantes matemáticas al cargar <cmath>
#include<cmath>
#include "complex.h"

using namespace std;

complex::complex () : _x(0.0) , _y(0.0) {}

complex::complex (const complex & c) : _x(c._x) , _y(c._y) {}

complex::complex (double a, double b): _x(a) , _y(b) {}

complex::~complex() {}

ostream& operator<<(ostream &os, const complex &c){
    os<<"("<<c._x<<","<<c._y<<")"<<endl;
    return os;
}

istream& operator>>(istream &is, complex &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;
}

complex& complex::operator=(const complex & b){
    this->_x = b._x;
    this->_y = b._y;
    return *this;
}

double complex::getReal()const {
    return this->_x;
}

double complex::getImag()const {
    return this->_y;
}

void complex::setReal(double xx){
    this->_x = xx;
}

void complex::setImag(double yy){
    this->_y = yy;
}

double complex::getArg()const {
    return std::sqrt(square(this->_x) + square(this->_y));
```

```
 74  }
 75
 76  double complex::getPha()const {
 77      return std::atan2(this->_y, this->_x);
 78  }
 79
 80
 81  /*|////////////| Operadores, solo los utilizados, faltan varios |\\\\\\\\\\\\|*/
 82
 83  const complex complex::operator+(const double f)const {
 84      return complex ( this->_x + f, this->_y );
 85  }
 86
 87  void complex::operator+=(const complex &c) {
 88      this->_x += c._x;
 89      this->_y += c._y;
 90  }
 91
 92  void complex::operator-=(const complex &c) {
 93      this->_x -= c._x;
 94      this->_y -= c._y;
 95  }
 96
 97
 98  /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
 99  /*|//////////////////| Operadores, operaciones binarias |\\\\\\\\\\\\\\\\\\|*/
100  /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
101  const complex complex::operator_add(const complex &w, const complex &z){
102      return complex ( w._x + z._x, w._y + z._y );
103  }
104
105  const complex complex::operator_subt(const complex &w, const complex &z){
106      return complex ( w._x - z._x, w._y - z._y );
107  }
108
109  // (x1+jy1)*(x2+jy2) = x1*x2 - y1*y2  + j( x1*y2 + y1*x2 )
110  const complex complex::operator_mult(const complex &w, const complex &z){
111      return complex ( w._x*z._x - w._y*z._y, w._x*z._y + w._y*z._x );
112  }
113
114  // (x1+jy1)/(x2+jy2) = ( x1*x2 + y1*y2 + j( y1*x2 - x1*y2 ) ) / ( x1^2 + y2^2 )
115  const complex complex::operator_div(const complex &w, const complex &z){
116      double aux = square(z._x) + square(z._y);
117
118      return complex ( (w._x * z._x + w._y * z._y) / aux,
119                       (w._y * z._x - w._x * z._y) / aux );
120  }
121
122  //w^z = exp(z * log(w))
123  const complex complex::operator_pow(const complex &w, const complex &z) {
124      return complex::exp( complex::operator_mult(z, complex::log(w)) );
125  }
126
127
128  /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
129  /*|//////////////////| Funciones, operaciones unarias |\\\\\\\\\\\\\\\\\\\\|*/
130  /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
131
132  //exp(z) = exp(x+jy) = exp(x) * exp(jy) = exp(x) * (cos(y) + j sin(y))
133  const complex complex::exp(const complex &z) {
134      return complex ( std::exp(z._x) * std::cos(z._y),
135                       std::exp(z._x) * std::sin(z._y) );
136  }
137
138  //ln(z) = ln|z| + j Phase(z)
139  const complex complex::log(const complex &z) {
140      return complex ( std::log(z.getArg()), z.getPha() );
141  }
142
143
144  // sin(z) = sin(x+jy) = sin(x) * cosh(y) + j cos(x) * sinh(y)
145  const complex complex::sin(const complex &z) {
146      return complex ( std::sin(z._x) * std::cosh(z._y),
```

```
147                     std::cos(z._x) * std::sinh(z._y) );
148 }
149
150 // cos(z) = cos(x+jy) = cos(x) * cosh(y) - j sin(x) * sinh(y)
151 const complex complex::cos(const complex &z) {
152     return complex ( std::cos(z._x) * std::cosh(z._y),
153                     -std::sin(z._x) * std::sinh(z._y) );
154 }
155
156 // Adaptaciones a devolución compleja de algunos métodos
157
158 const complex complex::real_as_complex(const complex &z)
159 {
160     return complex(z.getReal(), 0);
161 }
162
163 const complex complex::imag_as_complex(const complex &z)
164 {
165     return complex(z.getImag(), 0);
166 }
167
168 const complex complex::abs_as_complex(const complex &z)
169 {
170     return complex(z.getArg(), 0);
171 }
172
173 const complex complex::phase_as_complex(const complex &z)
174 {
175     return complex(z.getPha(), 0);
176 }
```

```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 _magic_number; // Número mágico PGM
    size_t              _width;        // Ancho del lienzo (imagen)
    size_t              _height;       // Alto del lienzo (imagen)
    pixel_t             _color_depth;  // 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
    static void _canvas_destroy(size_t, pixel_t **);


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() { _canvas_destroy(this->_height, this->_canvas); };

    // 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"

// Definición del número mágico
const string PGMimage::_magic_number("P2");
#define MAGIC_NUMBER_LENGTH 2



/*|////////////////////////////////|   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->_color_depth = 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->_color_depth = o._color_depth;

    // 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) // Tope, seguridad en altura
        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->_color_depth; }



```

```
/*|////////////////////////////////|   8)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|//////////////////// Cambio de profundidad de color |\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
void PGMimage::setColorDepth(pixel_t d)
{
    PGMimage::_validate_color_depth(d);

    // Cálculo de factor de escala y actualización de la profundidad
    float scale = (float) d / (float) this->_color_depth;
    this->_color_depth = d;

    // Cálculo de los datos con la nueva profundidad
    for (size_t i = 0; i < this->_height; i++)
        for (size_t j = 0; j < this->_width; j++)
            this->_canvas[i][j] *= scale;
}



/*|////////////////////////////////|   9)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|//////////////////////// Cambio de tamaño de imagen |\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
void PGMimage::resize(size_t w, size_t h)
{
    size_t w_max, h_max;
    pixel_t **auxcnv;

    // Memoria para el nuevo lienzo
    auxcnv = PGMimage::_new_canvas(w, h);

    // Copiado de los datos, con posible pérdida por recorte
    w_max = min(w, this->_width);
    h_max = min(h, this->_height);

    for (size_t i = 0; i < h_max; i++)
        for (size_t j = 0; j < w_max; j++)
            auxcnv[i][j] = this->_canvas[i][j];

    // Liberación de la memoria antigua
    PGMimage::_canvas_destroy(this->_height, this->_canvas);

    // Actualización de los valores
    this->_width  = w;
    this->_height = h;
    this->_canvas = auxcnv;
}
// NOTA: no se realiza un escalado de la imagen, solo se modifica el lienzo,
// resultando en posibles recortes o agregado de pixels con eventual basura.



/*|////////////////////////////////|  10)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////| Impresión en flujo/archivo/stdin |\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
ostream & operator<<(ostream &os, const PGMimage &c)
{
    // Encabezado del archivo
    os << c._magic_number << endl;
    os << c._width << ' ' << c._height << endl;
    os << (size_t) c._color_depth << endl;

    // Datos de pixels
    for (size_t i = 0; i < c._height; i++)
    {
        os << (size_t) c._canvas[i][0];
        for (size_t j = 1; j < c._width; j++)
            os << ' ' << (size_t) c._canvas[i][j];

        os << endl;
    }

    return os;
}
```

```
147
148
149
150 /*|//////////////////////////////|  11)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
151 /*|//////////////////| Carga desde flujo/archivo/stdin |\\\\\\\\\\\\\\\\\\|*/
152 /*|//////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
153 istream & operator>>(istream &is, PGMimage &c)
154 {
155     size_t w = 0, h = 0, aux = 0, i, j;
156     pixel_t d = 0, **auxcnv;
157     bool errors = true;
158     char mn[MAGIC_NUMBER_LENGTH + 1];
159     double scale = 1;
160
161     // Lectura del encabezado
162     is.get(mn, MAGIC_NUMBER_LENGTH + 1);
163     // Número mágico
164     if ( mn == c._magic_number )
165     {
166         PGMimage::_ignore_comments(is);
167         // Ancho y alto
168         if (is >> w && is >> h)
169         {
170             PGMimage::_ignore_comments(is);
171             // Profundidad de color
172             if (is >> aux && aux >= MIN_COLOR_DEPTH)
173             {
174                 errors = false;
175                 // Recorte en profundidad, de ser necesario
176                 if (aux > MAX_COLOR_DEPTH)
177                 {
178                     cerr << "Warning: max color depth is "
179                          << MAX_COLOR_DEPTH
180                          << ", the image will be adapted."
181                          << endl;
182                     // Escala para adaptar la imagen
183                     scale = (double) MAX_COLOR_DEPTH / (double) aux;
184                     d = MAX_COLOR_DEPTH;
185                 }
186                 else d = aux;
187             }
188         }
189     }
190
191     // Lectura de los datos de pixel
192     if (!errors)
193     {
194         // Memoria para el lienzo
195         auxcnv = PGMimage::_new_canvas(w, h);
196
197         // Carga de datos
198         for (i = 0; i < h; i++)
199         {
200             for (j = 0; j < w; j++)
201             {
202                 PGMimage::_ignore_comments(is);
203                 if (is >> aux) auxcnv[i][j] = scale * aux;
204                 else { errors = true; break; }
205             }
206             if (errors) break;
207         }
208
209         // Si no ha fallado la carga de valores
210         if (!errors)
211         {
212             // Actualización del objeto
213             PGMimage::_canvas_destroy(c._height, c._canvas);
214             c._canvas = auxcnv;
215             c._width = w;
216             c._height = h;
217             c._color_depth = d;
218         }
219         // En caso de falla, se deja el objeto intacto y se destruye auxcnv
```

```
220         else PGMimage::_canvas_destroy(h, auxcnv);
221     }
222
223     if (errors) // Si hubo errores, se indica en el stream
224         is.clear(ios::badbit);
225
226     return is;
227 }
228
229
230
231 /*|//////////////////////////////////|   *)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
232 /*|////////////////////////| Utilidades internas |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
233 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
234
235 // Ignorar comentarios estilo PGM en un stream
236 void PGMimage::_ignore_comments(istream &s)
237 {
238     char ch;
239     while (s >> ch)
240     {
241         if (ch == '#')
242         {
243             s.ignore(numeric_limits<streamsize>::max(), '\n');
244         }
245         else
246         {
247             s.putback(ch);
248             break;
249         }
250     }
251 }
252
253 // Pedir memoria para un lienzo de w x h
254 pixel_t** PGMimage::_new_canvas(size_t w, size_t h)
255 {
256     pixel_t **cnv = new pixel_t* [h];
257
258     for (size_t i = 0; i < h; i++)
259         cnv[i] = new pixel_t[w];
260
261     return cnv;
262 }
263
264 // Limitar profundidad de color
265 void PGMimage::_validate_color_depth(pixel_t &d)
266 {
267     if (d > MAX_COLOR_DEPTH) d = MAX_COLOR_DEPTH;
268     if (d < MIN_COLOR_DEPTH) d = MIN_COLOR_DEPTH;
269 }
270
271 // Destruir el lienzo sobre el objeto actual
272 void PGMimage::_canvas_destroy(size_t h, pixel_t **c)
273 {
274     for (size_t i = 0; i < h; i++)
275         delete [] c[i];
276
277     delete [] c;
278 }
```

```
 1  #ifndef NODE_H
 2  #define NODE_H
 3
 4  // Clases que utilizarán a la clase nodo
 5  template <class T> class queue;
 6  template <class T> class stack;
 7
 8
 9  template <class T>
10  class node {
11
12      public:
13          node(const T& v, node<T> *nxt = NULL);
14
15      private:
16          T _value;
17          node<T> *_next;
18
19      friend class stack<T>;
20      friend class queue<T>;
21  };
22
23  template <class T>
24  node<T>::node(const T& v, node<T> *nxt) {
25      _value = v;
26      _next = nxt;
27  }
28
29  #endif  /* NODE_H */
```

```
 1  #ifndef STACK_H
 2  #define STACK_H
 3
 4  #include <iostream>
 5  #include "node.h"
 6  #include "stack.h"
 7
 8  template <class T>
 9  class stack {
10
11      public:
12
13          stack() : _last(NULL) {};
14          ~stack();
15
16          bool isEmpty() const;
17          void push(const T& );
18          T pop();
19          T& topElement();
20
21      private:
22          node<T> *_last;
23
24  };
25
26  template <class T>
27  stack<T>::~stack() {
28
29     while(_last) pop();
30
31  }
32
33  template <class T>
34  void stack<T>::push(const T& v) {
35
36     node<T> *new_node;
37     new_node = new node<T>(v, _last);
38
39     _last = new_node; //asigno el nuevo nodo a la pila
40  }
41
42  template <class T>
43  T stack<T>::pop() {
44
45     node<T> *auxNode;
46     T v;
47
48     if(!_last) return T(); //pila vacía
49
50     auxNode = _last; //primer elemento de la pila
51
52     _last = auxNode->_next; //asignamos a la pila toda la pila sin el último nodo
53
54     v = auxNode->_value; //guardamos el valor del primero elemento de la pila
55
56     delete auxNode; //borramos el nodo
57
58     return v;
59  }
60
61
62  template <class T>
63  bool stack<T>::isEmpty() const {
64
65      return _last == NULL; //estar vacía es tener _last en NULL
66
67  }
68
69  template <class T>
70  T& stack<T>::topElement() {
71
72      return _last->_value;
73
```

```
74  }
75
76  #endif  /* STACK_H */
```

```
 1  #ifndef QUEUE_H
 2  #define QUEUE_H
 3
 4  #include <iostream>
 5  #include "node.h"
 6
 7  template <class T>
 8  class queue {
 9      public:
10          queue() : _first(NULL), _last(NULL) {}
11          ~queue();
12
13          void enqueue(const T&);
14          T dequeue();
15          bool isEmpty() const;
16          T& frontElement();
17          const T& lastAdded() const;
18
19      private:
20          node<T> *_first, *_last;
21
22  };
23
24
25  template <class T>
26  queue<T>::~queue() {
27
28     while(_first) dequeue();
29
30  }
31
32  template <class T>
33  void queue<T>::enqueue(const T& v){
34
35      node<T> *newNode;
36
37      newNode = new node<T>(v); // creamos un nuevo auxNode
38
39      // Si la cola no estaba vacía, añadimos el nuevo a continuación del último
40      if(_last) _last->_next = newNode;
41
42      _last = newNode; // Ahora, el último elemento de la cola es el nuevo auxNode
43
44      // Si la cola estaba vacía, ahora el primero también es el nuevo auxNode
45      if(!_first) _first = newNode;
46
47  }
48
49  template <class T>
50  T queue<T>::dequeue(){
51
52      node<T> *auxNode;
53      T v;
54
55      auxNode = _first;
56      if(!auxNode) return T();
57
58      _first = auxNode->_next; //asignamos al primero el segundo nodo
59
60      v = auxNode->_value;
61
62      delete auxNode;
63
64      //si la cola quedó vacía, ultimo debe ser NULL también
65      if(!_first) _last = NULL;
66
67      return v;
68
69  }
70
71  template <class T>
72  bool queue<T>::isEmpty() const{
73
```

```
74      return _first == NULL; //estar vacía es tener _first en NULL
75
76  }
77
78  template <class T>
79  T& queue<T>::frontElement(){
80
81      return _first->_value;
82
83  }
84
85
86  template <class T>
87  const T& queue<T>::lastAdded() const{
88
89      return _last->_value;
90
91  }
92
93  #endif  /* QUEUE_H */
```

```cpp
#ifndef UTILS_H
#define UTILS_H

#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <string>
#include "cmdline.h"
#include "stack.h"
#include "queue.h"
#include "complex.h"
#include "parser.h"

using namespace std;

////////////////////// Variables globales de main.cpp //////////////////////
extern option_t     options_[];        // Opciones CLA
extern istream      *iss_;             // Puntero a stream de entrada
extern ostream      *oss_;             // Puntero a stream de salida
extern fstream      ifs_;              // Archivo de entrada
extern fstream      ofs_;              // Archivo de salida
extern char         *prog_name_;       // Nombre del programa
extern double       map_w_;            // Ancho de la región de mapeo
extern double       map_h_;            // Alto de la región de mapeo
extern complex      map_c_;            // Centro de la región de mapeo
extern stack<token> rpn_expr_;         // Expresión convertida a RPN



/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||| Utilidades ||||||||||||||||||||||||||||||||*/
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/

// 1) CLA: Archivo de entrada
void opt_input(const string &);

// 2) CLA: Archivo de salida
void opt_output(const string &);

// 3) CLA: Función a aplicar
void opt_function(const string &);

// 4) CLA: Región del plano complejo
void opt_region(const string &);

// 5) CLA: Ayuda
void opt_help(const string &);

// 6) Obtener complejo asociado a los índices
void get_complex_from_index(complex &, size_t, size_t, size_t, size_t);

// 7) Obtener la fila asociada al complejo ( [i][ ] )
void get_row_from_complex(size_t &, const complex &, size_t);

// 8) Obtener la columna asociada al complejo ( [ ][j] )
void get_col_from_complex(size_t &, const complex &, size_t);


#endif    /* UTILS_H */
```

```cpp
#include "utils.h"


/*|////////////////////////////////|   1)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|//////////////////////| CLA: Archivo de entrada |\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
void opt_input(const string &arg)
{
    // Por defecto stdin, o bien archivo
    if (arg == "-")
    {
        iss_ = &cin;
    }
    else
    {
        ifs_.open(arg.c_str(), ios::in);
        iss_ = &ifs_;
    }

    // Comprobación de errores
    if ( !iss_->good() )
    {
        cerr << "Cannot open "
             << arg
             << "."
             << endl;
        exit(1);
    }
}



/*|////////////////////////////////|   2)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|//////////////////////| CLA: Archivo de salida |\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
void opt_output(const string &arg)
{
    // Por defecto stdout, o bien archivo
    if (arg == "-")
    {
        oss_ = &cout;
    }
    else
    {
        ofs_.open(arg.c_str(), ios::out);
        oss_ = &ofs_;
    }

    // Comprobación de errores
    if ( !oss_->good() )
    {
        cerr << "Cannot open "
             << arg
             << "."
             << endl;
        exit(1);
    }
}



/*|////////////////////////////////|   3)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|//////////////////////| CLA: Función a aplicar |\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
void opt_function(const string &arg)
{
    queue<token> tokenized_expr;

    parse_expression_in_tokens(arg, tokenized_expr);
    convert_to_RPN(rpn_expr_, tokenized_expr);
}


```

```
 74
 75 /*|////////////////////////////////|   4)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
 76 /*|////////////////////| CLA: Región del plano complejo |\\\\\\\\\\\\\\\\\\\\|*/
 77 /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
 78 void opt_region(const string &arg)
 79 {
 80     double aux;
 81     stringstream arg_stream(arg);
 82     bool errors = true;
 83
 84     // Lectura de los parámetros, ignorando el separador (',' o cualquier char)
 85     if (arg_stream >> map_w_)
 86     {
 87         arg_stream.ignore();
 88
 89         if (arg_stream >> map_h_)
 90         {
 91             arg_stream.ignore();
 92
 93             if (arg_stream >> aux)
 94             {
 95                 map_c_.setReal(aux);
 96                 arg_stream.ignore();
 97
 98                 if (arg_stream >> aux)
 99                 {
100                     map_c_.setImag(aux);
101
102                     // Si se llegó hasta aquí, se pudieron leer los
103                     // 4 parámetros, si hay algo más se ignora
104                     errors = false;
105                 }
106             }
107         }
108     }
109
110     // Error de lectura, región inválida
111     if (errors)
112     {
113         cerr << "Invalid region description: "
114              << arg
115              << "."
116              << endl;
117         exit(1);
118     }
119
120     // Error por ancho o alto no "positivos distintos de cero"
121     if (map_w_ <= 0 || map_h_ <=0)
122     {
123         cerr << map_w_ << "," << map_h_
124              << ": must be positive nonzero numbers."
125              << endl;
126         exit(1);
127     }
128
129 }
130
131
132
133 /*|////////////////////////////////|   5)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
134 /*|//////////////////////////////| CLA: Ayuda |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
135 /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
136 void opt_help(const string &arg)
137 {
138     cout << "Usage: "
139          << prog_name_
140          << " [-i file] [-o file] [-r w,h,x0,y0] [-f expression(z)]"
141          << endl;
142     exit(0);
143 }
144
145
146
```

```
147 /*|////////////////////////////////|   6)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
148 /*|///////////////| Obtener complejo asociado a los índices |\\\\\\\\\\\\\\\|*/
149 /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
150 void get_complex_from_index(complex &z, size_t i, size_t j, size_t h, size_t w)
151 {
152     if ( h && w && i < h && j < w)
153     {
154         z.setReal( map_w_ * ( (j + 0.5)/w  -  0.5 ) );
155         z.setImag( map_h_ * ( 0.5  -  (i + 0.5)/h ) );
156         z += map_c_;
157     }
158 }
159
160
161
162 /*|////////////////////////////////|   7)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
163 /*|////////////| Obtener la fila asociada al complejo ( [i][ ] ) |\\\\\\\\\\\\|*/
164 /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
165 void get_row_from_complex(size_t &row, const complex &z, size_t h)
166 {
167     row = h * ( 0.5 - (z.getImag()-map_c_.getImag())/map_h_ );
168 }
169
170
171
172 /*|////////////////////////////////|   8)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
173 /*|///////////| Obtener la columna asociada al complejo ( [ ][j] ) \\\\\\\\\\|*/
174 /*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
175 void get_col_from_complex(size_t &col, const complex &z, size_t w)
176 {
177     col = w * ( 0.5 + (z.getReal()-map_c_.getReal()) / map_w_ );
178 }
```

```cpp
#ifndef PARSER_H
#define PARSER_H

#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdlib>
#define _USE_MATH_DEFINES // Constantes matemáticas al cargar <cmath>
#include <cmath>
#include <string>
#include "queue.h"
#include "stack.h"
#include "complex.h"

using namespace std;


#define NOT_FOUND  -1

// Macros de función para cómoda detección de casos
#define is_number_start(s) ( isdigit((s)[0]) || (s) == "." )
#define is_parenthesis(s)  ( (s) == "(" || (s) == ")" )
#define is_operator(s)     ( find_in_list(operator_tokens_, s) != NOT_FOUND )
#define is_special(s)      ( find_in_list(special_tokens_,  s) != NOT_FOUND )
#define is_function(s)     ( find_in_list(function_tokens_, s) != NOT_FOUND )

// Chequeo de +/- como operador unario: si es el primero o viene luego de '('
#define check_for_unary_op(q) ((q).isEmpty() || \
                               (q).lastAdded().isOpenParenthesis())

// Punteros a función para las operaciones
typedef const complex (*operator_t)(const complex &, const complex &);
typedef const complex (*function_t)(const complex &);



/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||||||||||| Token ||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/

#define TOKEN_IS_VALUE "0v0"

class token
{
    double _v; // Valor, vale NaN si no es del tipo 'valor'
    string _s; // Cadena del token, vale TOKEN_IS_VALUE si es del tipo 'valor'


public:
    // 1) Constructores
    token(const double &v = 0); // De un token del tipo 'valor', 0 por defecto
    token(const string &);      // De un token que no es del tipo 'valor'
    token(const token &);       // Por copia

    // 2) Obtener como string (devuelve número en string si es del tipo 'valor')
    string getAsString() const;

    // 3) Obtener como double (devuelve NAN, si no es del tipo 'valor')
    double getAsDouble() const;

    // 4) Booleanos is...
    bool isValue() const;             // ¿Es del tipo 'valor'?
    bool isParenthesis() const;       // ¿Es un paréntesis?
    bool isOpenParenthesis() const;   // ¿Es un paréntesis abierto?
    bool isClosedParenthesis() const; // ¿Es un paréntesis cerrado?
    bool isOperator() const;          // ¿Es un operador?
    bool isSpecial() const;           // ¿Es especial?
    bool isFunction() const;          // ¿Es una función?

    // 5) Precedencia, si es que el token es un operador o función
    int precedence() const;

    // 6) Impresión en flujo/archivo/stdin
```

```
74      friend ostream & operator<<(ostream &, const token &);
75  };
76
77
78
79  /*|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
80  /*||||||| Variables especiales para parseo y evaluación de la expresión ||||||*/
81  /*|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
82
83
84  /*|/////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
85  /*|/////////////| Funciones a interpretar, operaciones unarias |\\\\\\\\\\\\\|*/
86  /*|/////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
87
88  // Cadenas asociadas, el orden importa
89  const string function_tokens_[] =
90  {
91      "exp",
92      "ln",
93      "sin",
94      "cos",
95      "re",
96      "im",
97      "abs",
98      "phase",
99  // No olvidar centinela de cadena vacía
100     ""
101 };
102
103 // Punteros a funciones asociados, el orden importa
104 const function_t function_pointers_[] =
105 {
106     complex::exp,
107     complex::log,
108     complex::sin,
109     complex::cos,
110     complex::real_as_complex,
111     complex::imag_as_complex,
112     complex::abs_as_complex,
113     complex::phase_as_complex
114 };
115
116 /*|/////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
117 /*|////////////| Operadores a interpretar, operaciones binarias |\\\\\\\\\\\\|*/
118 /*|/////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
119
120 // Cadenas asociadas, el orden importa
121 const string operator_tokens_[] =
122 {
123     "+",
124     "-",
125     "*",
126     "/",
127     "^",
128 // No olvidar centinela de cadena vacía
129     ""
130 };
131
132 // Punteros a funciones asociados, el orden importa
133 const operator_t operator_pointers_[] =
134 {
135     complex::operator_add,
136     complex::operator_subt,
137     complex::operator_mult,
138     complex::operator_div,
139     complex::operator_pow
140 };
141
142
143 /*|/////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
144 /*|////////////////////| Tokens especiales a interpretar |\\\\\\\\\\\\\\\\\\\\|*/
145 /*|/////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
146
```

```
#define PIXEL_TOKEN "z" // Token simbólico que representa el pixel a transformar
// Cadenas asociadas, el orden importa, PIXEL_TOKEN debe ir último
const string special_tokens_[] =
{
    "j",
    "e",
    "pi",
    PIXEL_TOKEN,
// No olvidar centinela de cadena vacía
    ""
};

// Complejos asociados a tokens especiales (excepto z), el orden importa
const complex special_complex_[] =
{
    complex(0, 1),       // j
    complex(M_E, 0),     // e
    complex(M_PI, 0)     // pi
};



/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||| Utilidades ||||||||||||||||||||||||||||||||*/
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/

// 1) Función para buscar en una lista con centinela de cadena vacía
int find_in_list(const string [], const string &);

// 2) Función para parsear la expresión de entrada partiéndola en tokens
void parse_expression_in_tokens(const string &, queue<token> &);

// 3) Conversión a notación polaca inversa
void convert_to_RPN(stack<token> &, queue<token> &);
void error_handler_unexpected_token(const token &);
void error_handler_mismatched_parentheses();


#endif /* PARSER_H */
```

```cpp
#include "parser.h"


/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||||| Token |||||||||||||||||||||||||||||||||*/
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/


/*|////////////////////////////////|   1)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////| Constructores |\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/

// De un token del tipo 'valor', por defecto (tipo 'valor' con valor 0)
token::token(const double &v)
{
    this->_v = v;
    this->_s = TOKEN_IS_VALUE;
}

// De un token que no es del tipo 'valor'
token::token(const string &s)
{
    this->_v = NAN;
    this->_s = s;
}

// Por copia
token::token(const token &t)
{
    this->_v = t._v;
    this->_s = t._s;
}



/*|////////////////////////////////|   2)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|/////////////////////////| Obtener como string |\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
string token::getAsString() const
{
    if (this->isValue())
    {
        ostringstream aux;
        aux << this->_v;
        return aux.str();
    }

    return this->_s;
}
// NOTA: si bien es capaz de devolver el valor en una string, no utilizar
// innecesariamente.



/*|////////////////////////////////|   3)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|/////////////////////////| Obtener como double |\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
double token::getAsDouble() const
{
    return this->_v;
}



/*|////////////////////////////////|   4)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|///////////////////////////| Booleanos is... |\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/

// ¿Es del tipo 'valor'?
bool token::isValue() const
{
    return this->_s == TOKEN_IS_VALUE;
}
```

```
 74
 75  // ¿Es un paréntesis?
 76  bool token::isParenthesis() const
 77  {
 78      if (this->isValue()) return false;
 79      return is_parenthesis(this->_s);
 80  }
 81
 82  // ¿Es un paréntesis abierto?
 83  bool token::isOpenParenthesis() const
 84  {
 85      if (this->isValue()) return false;
 86      return this->_s == "(";
 87  }
 88
 89  // ¿Es un paréntesis cerrado?
 90  bool token::isClosedParenthesis() const
 91  {
 92      if (this->isValue()) return false;
 93      return this->_s == ")";
 94  }
 95
 96  // ¿Es un operador?
 97  bool token::isOperator() const
 98  {
 99      if (this->isValue()) return false;
100      return is_operator(this->_s);
101  }
102
103  // ¿Es especial?
104  bool token::isSpecial() const
105  {
106      if (this->isValue()) return false;
107      return is_special(this->_s);
108  }
109
110  // ¿Es una función?
111  bool token::isFunction() const
112  {
113      if (this->isValue()) return false;
114      return is_function(this->_s);
115  }
116
117
118
119  /*|//////////////////////////////|   5)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
120  /*|///////| Precedencia, si es que el token es un operador o función |\\\\\\\|*/
121  /*|//////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
122  int token::precedence() const
123  {
124      if (this->isOperator())
125      {
126          if (this->_s == "+" || this->_s == "-") return 0;
127          if (this->_s == "*" || this->_s == "/") return 1;
128          if (this->_s == "^") return 3;
129      }
130
131      if (this->isFunction()) return 2;
132
133      return NOT_FOUND; // Si la precedencia no es aplicable, devuelve NOT_FOUND
134  }
135
136
137
138  /*|//////////////////////////////|   6)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
139  /*|//////////////////| Impresión en flujo/archivo/stdin |\\\\\\\\\\\\\\\\\\|*/
140  /*|//////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
141  ostream & operator<<(ostream &os, const token &t)
142  {
143      if (t.isValue()) os << t._v;
144      else             os << t._s;
145
146      return os;
```

```
147 }
148
149
150
151
152 /*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
153 /*|||||||||||||||||||||||||||||||| Utilidades ||||||||||||||||||||||||||||||||||||||||||||||||||*/
154 /*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
155
156
157 /*|//////////////////////////////|   1)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
158 /*|////| Función para buscar en una lista con centinela de cadena vacía |\\\\|*/
159 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
160 int find_in_list(const string l[], const string &s)
161 {
162     for (size_t i = 0; !l[i].empty(); i++)
163         if (l[i] == s)
164             return i; // Si lo encuentra devuelve su posición
165
166     return NOT_FOUND; // Si no lo encuentra se devuelve NOT_FOUND
167 }
168
169
170
171 /*|//////////////////////////////|   2)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
172 /*|//| Función para parsear la expresión de entrada partiéndola en tokens |\\|*/
173 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
174 void parse_expression_in_tokens(const string &input, queue<token> &output)
175 {
176     stringstream expr(input);
177     string aux_s;
178     double aux_n;
179
180
181     while (expr)
182     {
183         // Se intenta cargar un carácter en aux_s
184         aux_s = expr.peek();
185
186         // Si es un espacio, se ignora y se vuelve a empezar
187         if ( isblank(aux_s[0]) )
188         {
189             expr.ignore();
190             continue;
191         }
192
193         // Si es el comienzo de un número, se intenta leerlo
194         if ( is_number_start(aux_s) )
195         {
196             // Si no se logra, se sale con un error
197             if ( !(expr >> aux_n) )
198             {
199                 cerr << "Error: wrong input number in the expression."
200                      << endl;
201                 exit(1);
202             }
203
204             // Si se logra, se agrega a la cola de salida
205             output.enqueue(token(aux_n));
206         }
207
208         // Si no, puede tratarse de un operador o un paréntesis
209         else if ( is_operator(aux_s) || is_parenthesis(aux_s) )
210         {
211             // Ya es algo válido, se elimina del stream
212             expr.ignore();
213
214             // Caso especial, "-" como operador unario, se agrega un 0 antes
215             if ( aux_s == "-" && check_for_unary_op(output) )
216                 output.enqueue(token(0));
217
218             // Caso especial, "+" como operador unario, se ignora
219             if ( aux_s == "+" && check_for_unary_op(output) )
```

```
220                 continue;
221
222             output.enqueue(token(aux_s));
223         }
224
225         // O si no, queda el grupo alfabético de output: funciones o especiales
226         else
227         {
228             // Se levantan todos los caracteres alfabéticos a la cadena auxiliar
229             expr.ignore(); // El primero ya está en la cadena, se ignora
230             while ( isalpha(expr.peek()) ) aux_s += expr.get();
231
232             // Si coincide con alguno de estos, se encola
233             if ( is_function(aux_s) || is_special(aux_s) )
234             {
235                 output.enqueue(token(aux_s));
236             }
237
238             // Si no es así, y tampoco se terminó la entrada, hay un error
239             else if (expr)
240             {
241                 cerr << "Error: malformed expression near of: "
242                      << aux_s
243                      << "."
244                      << endl;
245                 exit(1);
246             }
247         }
248     }
249 }
250
251
252
253 /*|//////////////////////////////|   3)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
254 /*|//////////////////| Conversión a notación polaca inversa |\\\\\\\\\\\\\\\\\\|*/
255 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
256 void convert_to_RPN(stack<token> &result, queue<token> &tokens){
257
258     token tok;
259     stack<token> aux; // Pila auxiliar para la conversión
260
261     bool expect_operator_flag = false;
262     bool expect_number_flag = false;
263     bool expect_function_flag = true;
264
265     while (!tokens.isEmpty()) {
266
267         tok = tokens.dequeue();
268
269         //Si el token es un operador, o1, entonces:
270         if (tok.isOperator()){
271
272             if (!expect_operator_flag)
273                 error_handler_unexpected_token(tok);
274
275             /*
276             mientras que haya un operador, o2, en el tope de la pila (esto
277             excluye el paréntesis abierto), y
278                 * o1 es asociativo izquierdo y su precedencia es menor que (una
279                 precedencia más baja) o igual a la de o2, ó
280                 * o1 es asociativo derecho y su precedencia es menor que (una
281                 precedencia más baja) que la de o2,
282             retire (pop) de la pila el o2, y póngalo en la cola de salida.
283             */
284             while ( !aux.isEmpty() &&
285                     (  aux.topElement().isOperator() ||
286                        aux.topElement().isFunction()  )   &&
287                      aux.topElement().precedence() >= tok.precedence() ) {
288
289                 result.push(aux.topElement());
290                 aux.pop();
291             }
292
```

```
293             //ponga (push) o1 en el tope de la pila.
294             aux.push(tok);
295 
296             expect_operator_flag = false;
297             expect_function_flag = true;
298             expect_number_flag = true;
299 
300         } else if (tok.isFunction()){
301 
302             if (!expect_function_flag)
303                 error_handler_unexpected_token(tok);
304 
305             while ( !aux.isEmpty() && aux.topElement().isOperator() &&
306                     aux.topElement().precedence() >= tok.precedence() ) {
307 
308                 result.push(aux.topElement());
309                 aux.pop();
310             }
311 
312             //ponga (push) o1 en el tope de la pila.
313             aux.push(tok);
314 
315             expect_operator_flag = false;
316             expect_function_flag = false;
317             expect_number_flag = false;
318 
319         //Si el token es un paréntesis abierto, entonces póngalo en la pila.
320         } else if (tok.isOpenParenthesis()) {
321 
322             //si esperaba un operador
323             if (expect_operator_flag)
324                 error_handler_unexpected_token(tok);
325 
326             aux.push(tok);
327 
328             expect_number_flag = false;
329 
330         //Si el token es un paréntesis derecho
331         } else if (tok.isClosedParenthesis()) {
332 
333             //esto debe aparecer después de un numero/función, no de un operador
334             if (!expect_operator_flag)
335                 error_handler_unexpected_token(tok);
336 
337             /*Hasta que el token en el tope de la pila sea un paréntesis
338             abierto, retire (pop) a los operadores de la pila y colóquelos
339             en la cola de salida.*/
340             while ( !aux.isEmpty() &&
341                     !aux.topElement().isOpenParenthesis() ) {
342 
343                 result.push(aux.topElement());
344                 aux.pop();
345             }
346 
347             //Si la pila se termina sin encontrar un paréntesis abierto,
348             //entonces hay paréntesis sin pareja.
349             if (aux.isEmpty())
350                 error_handler_mismatched_parentheses();
351 
352             //Retire (pop) el paréntesis abierto de la pila,
353             //pero no lo ponga en la cola de salida.
354             aux.pop();
355 
356             /* Ahora esperamos un operador */
357             expect_operator_flag = true;
358             expect_function_flag = true;
359             expect_number_flag = false;
360 
361         //encuentre un numero
362         } else if (tok.isValue() || tok.isSpecial()) {
363 
364             /* If we're expecting an operator, we're very disappointed. */
365             if (expect_operator_flag && !expect_number_flag)
```

```
366                 error_handler_unexpected_token(tok);
367
368             //Si el token es un número, se agrega a la cola de salida
369             result.push(tok);
370
371             expect_operator_flag = true;
372             expect_function_flag = false;
373             expect_number_flag = false;
374
375         }
376     }
377
378     //ya se parsearon todos los tokens. Esperamos un operador
379     //ya que lo ultimo fue un valor
380     if (!expect_operator_flag)
381         error_handler_unexpected_token(tok);
382
383     /*
384     Cuando no hay más tokens para leer:
385         Mientras todavía haya tokens de operadores en la pila:
386             Si el token del operador en el tope de la pila es un paréntesis,
387             entonces hay paréntesis sin la pareja correspondiente.
388             retire (pop) al operador y póngalo en la cola de salida.
389     */
390     while (!aux.isEmpty()) {
391         if (aux.topElement().isOpenParenthesis())
392             error_handler_mismatched_parentheses();
393
394         result.push(aux.topElement());
395         aux.pop();
396     }
397
398 }
399
400 // Extra: manejador de error en caso de token inesperado
401 void error_handler_unexpected_token(const token &t) {
402     cerr << "Error: invalid syntax in the expression, near token: "
403          << t
404          << "."
405          << endl;
406     exit(1);
407 }
408
409 // Extra: manejador de error en caso de paréntesis desbalanceado
410 void error_handler_mismatched_parentheses() {
411     cerr << "Error: mismatched parentheses in the expression."
412          << endl;
413     exit(1);
414 }
```

```
#ifndef OPTREE_H
#define OPTREE_H

#include <iostream>
#include <cstdlib>
#define _USE_MATH_DEFINES // Constantes matemáticas al cargar <cmath>
#include <cmath>
#include "complex.h"
#include "stack.h"
#include "parser.h"

using namespace std;


// Tipo de nodo en el árbol de operaciones
typedef enum
{
    NODE_UNARY_OP,
    NODE_BINARY_OP,
    NODE_STATIC_COMPLEX,
    NODE_DYNAMIC_COMPLEX,
    NODE_PIXEL_COMPLEX,
    NODE_UNDEFINED
} node_type;



/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||| OptreeNode ||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
class optree_node {

    node_type     _t;      // Tipo de nodo
    complex const *_c;     // Puntero al complejo para evaluar
    operator_t    _bin_op; // Puntero a la operación binaria
    function_t    _un_op;  // Puntero a la operación unaria
    optree_node   *_left;  // Subárbol izquierdo (o hijo en operaciones unarias)
    optree_node   *_right; // Subárbol derecho
    optree_node   *_up;    // Padre


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

    // Simplificar el sub-árbol eliminando las expresiones independientes de z
    // NOTA: devuelve true si el subárbol depende de z
    bool simplify();


public:
    // 1) Constructor por defecto
    optree_node();

    // 2) Constructor a partir de token
    optree_node(const token &, optree_node *);

    // 3) Destructor
    ~optree_node();

    // 4) Operar, para realizar la operación
    const complex operate(const complex *) const;

    friend class optree;

private:
    // TODO: constructor por copia
    optree_node(const optree_node &);
};




/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||| Optree ||||||||||||||||||||||||||||||||||||||||||||||*/
```

```
74  /*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
75  class optree {
76
77      optree_node *_root;       // Raíz del árbol
78      complex     *_asoc_pixel; // Puntero al complejo asociado al pixel
79
80  public:
81      // 1) Constructor por defecto
82      optree() { _root = NULL; _asoc_pixel = NULL; }
83
84      // 2) Constructor desde pila de tokens con RPN + complejo de pixel
85      optree(stack<token> &, complex &);
86
87      // 3) Destructor
88      ~optree() { delete this->_root; }
89
90      // 4) Operar, para realizar la operación
91      const complex operate() const;
92
93  private:
94      // TODO: constructor por copia
95      optree(const optree &);
96  };
97
98
99  #endif // OPTREE_H
```

```cpp
#include "optree.h"


/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||| OptreeNode ||||||||||||||||||||||||||||||||*/
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/

/*|/////////////////////////////////|   1)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|////////////////////////| Constructor por defecto |\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|/////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
optree_node::optree_node()
{
    this->_t = NODE_UNDEFINED;
    this->_c       = NULL;
    this->_un_op   = NULL;
    this->_bin_op  = NULL;
    this->_left    = NULL;
    this->_right   = NULL;
    this->_up      = NULL;
}



/*|/////////////////////////////////|   2)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
/*|//////////////////////| Constructor a partir de token |\\\\\\\\\\\\\\\\\\\|*/
/*|/////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
optree_node::optree_node(const token &tok, optree_node *father = NULL)
{
    /*
        Los tipos de token son:

            *       value: tienen un double y aquí pasan a la parte real de un
                           complejo dinámico. Tipo: NODE_DYNAMIC_COMPLEX.

            * parenthesis: ya no hay de este tipo luego de pasar a RPN, se
                           imprimirá un error.

            *    operator: serán asociados a operaciones binarias.
                           Tipo: NODE_BINARY_OP.

            *    function: serán asociados a operaciones unarias.
                           Tipo: NODE_UNARY_OP.

            *     special:
                        -        z: será asociado al complejo de pixel.
                                    Tipo: NODE_PIXEL_COMPLEX.

                        - j, e, pi: serán asociados a complejos miembros
                                    estáticos del array special_complex_.
                                    Tipo: NODE_STATIC_COMPLEX.
    */

    if (tok.isParenthesis())
    {
        cerr << "Internal Error: the token can't be a parenthesis, "
             << "RPN convert is required before to make the optree."
             << endl;
        exit(2);
    }

    // Operandos
    if (tok.isSpecial() || tok.isValue())
    {
        this->_un_op  = NULL;
        this->_bin_op = NULL;

        // Special Token
        if (tok.isSpecial())
        {
            // Si es PIXEL_TOKEN (es decir, z)
            if (tok.getAsString() == PIXEL_TOKEN)
            {
                this->_t = NODE_PIXEL_COMPLEX;
```

```
74                  this->_c = NULL;
75              }
76              else // Si no es z, se trata de un complejo estático
77              {
78                  this->_t = NODE_STATIC_COMPLEX;
79                  this->_c = &special_complex_[
80                             find_in_list(special_tokens_, tok.getAsString()) ];
81              }
82
83          }
84          // Value Token
85          else
86          {
87             this->_t = NODE_DYNAMIC_COMPLEX;
88             this->_c = new complex(tok.getAsDouble(), 0);
89          }
90      }
91
92      // Operaciones
93      if (tok.isOperator() || tok.isFunction())
94      {
95          this->_c = NULL;
96
97          // Operator Token -> Operación binaria
98          if (tok.isOperator())
99          {
100             this->_t = NODE_BINARY_OP;
101             this->_un_op = NULL;
102             this->_bin_op = operator_pointers_[
103                     find_in_list(operator_tokens_, tok.getAsString()) ];
104         }
105
106         // Function Token -> Operación unaria
107         else
108         {
109             this->_t = NODE_UNARY_OP;
110             this->_bin_op = NULL;
111             this->_un_op = function_pointers_[
112                     find_in_list(function_tokens_, tok.getAsString()) ];
113         }
114     }
115
116
117     this->_left  = NULL;
118     this->_right = NULL;
119     this->_up = father;
120 }
121
122
123
124 /*|//////////////////////////////////|   3)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
125 /*|//////////////////////////////////| Destructor |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
126 /*|////////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
127 optree_node::~optree_node()
128 {
129     if (this->_t == NODE_DYNAMIC_COMPLEX) delete this->_c;
130     if (this->_left != NULL)              delete this->_left;
131     if (this->_right != NULL)             delete this->_right;
132 }
133
134
135
136 /*|//////////////////////////////////|   4)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
137 /*|//////////////////| Operar, para realizar la operación |\\\\\\\\\\\\\\\\\\|*/
138 /*|////////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
139 const complex optree_node::operate(const complex *z) const
140 {
141     // z es un puntero que apunta al complejo asociado al pixel para operar
142
143     // Caso base y de corte, operandos.
144     if ( this->_t == NODE_STATIC_COMPLEX || this->_t == NODE_DYNAMIC_COMPLEX )
145         return *(this->_c);
146
```

```
147      if ( this->_t == NODE_PIXEL_COMPLEX )
148          return *z;
149
150      // Operación binaria (operadores)
151      if ( this->_t == NODE_BINARY_OP )
152          return this->_bin_op(this->_left->operate(z), this->_right->operate(z));
153
154      // Operación unaria (funciones)
155      if  (this->_t == NODE_UNARY_OP )
156          return this->_un_op(this->_left->operate(z));
157
158      // Si se llega hasta aquí, el nodo estaba sin definir, error
159      cerr << "Internal Error: there are some node, not setted in the optree."
160           << endl;
161      exit(2);
162  }
163
164
165
166  /*|//////////////////////////////|   *)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
167  /*|////////////////////////| Utilidades internas |\\\\\\\\\\\\\\\\\\\\\\\\|*/
168  /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
169
170  // Simplificar el sub-árbol eliminando las expresiones independientes de z
171  // NOTA: devuelve true si el subárbol depende de z
172  bool optree_node::simplify()
173  {
174      bool pixel_dependent;
175
176      // Caso base: hoja, entonces depende de él, si es z u otra cosa
177      if (this->_left == NULL && this->_right == NULL)
178          return this->_t == NODE_PIXEL_COMPLEX;
179
180      // Si tiene único hijo (está a la izquierda, op unaria), depende de éste
181      if (this->_right == NULL)
182          pixel_dependent = this->_left->simplify();
183
184      /*
185         Si tiene ambos hijos (op binaria), con que uno dependa de z, suficiente.
186         Pero hay que visitar ambos hijos con simplify(). El operador || no evalúa
187         si ya encontró true a izquierda, por eso se utilizan dos líneas, ubicando
188         en la segunda, el llamado a simplify() a la izquierda.
189      */
190      else
191      {
192          pixel_dependent = this->_left->simplify();
193          pixel_dependent = this->_right->simplify() || pixel_dependent;
194      }
195
196      // Si no es pixel dependiente, como no es hoja (caso base), se simplifica
197      if (!pixel_dependent)
198      {
199          /*
200             El resultado no depende de z, por eso no hay problema con pasar NULL
201             a operate(). Además, como el nodo actual no es hoja, es una
202             operación, por lo tanto _c está libre (en NULL).
203          */
204          this->_c = new complex(this->operate(NULL));
205
206          // Se cambia el tipo del nodo subárbol, ahora convertido en hoja
207          this->_t = NODE_DYNAMIC_COMPLEX;
208          this->_un_op  = NULL;
209          this->_bin_op = NULL;
210
211          // Se destruye el hijo izquierdo (tiene que existir)
212          delete this->_left;
213          this->_left = NULL;
214          // Si tiene, se destruye el hijo derecho
215          if (this->_right != NULL)
216          {
217              delete this->_right;
218              this->_right = NULL;
219          }
```

```
220 #ifdef DEBUG
221         cerr << "    Construyendo árbol: simplificación realizada.\n";
222 #endif
223     }
224 
225     return pixel_dependent;
226 }
227 
228 
229 
230 
231 /*|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
232 /*|||||||||||||||||||||||||||||||||| Optree ||||||||||||||||||||||||||||||||||||||||||||*/
233 /*|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
234 #define is_operation(t) ((t) == NODE_UNARY_OP || (t) == NODE_BINARY_OP)
235 
236 /*|//////////////////////////////|   2)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
237 /*|/////| Constructor desde pila de tokens con RPN + complejo de pixel |\\\\\|*/
238 /*|//////////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
239 optree::optree(stack<token> &rpn, complex &z)
240 {
241     optree_node *current;
242 
243     // Primer token a la raíz
244     this->_root = new optree_node(rpn.pop());
245 
246     // current siempre será una operación, por la construcción del algoritmo,
247     // si no lo fuera, la pila ya está vacía.
248     current = this->_root;
249     while (!rpn.isEmpty())
250     {
251         // Si actual es unario
252         if (current->_t == NODE_UNARY_OP)
253         {
254             // Único hijo libre, se usa _left
255             if (current->_left == NULL)
256             {
257                 // Token al hijo
258                 current->_left = new optree_node(rpn.pop(), current);
259 
260                 // Si el token era una operación, bajar
261                 if (is_operation(current->_left->_t))
262                     current = current->_left;
263             }
264             // Hijo ocupado, subir
265             else
266                 current = current->_up;
267         }
268         // Si actual es binario
269         else
270         {
271             // Derecha libre
272             if (current->_right == NULL)
273             {
274                 // Token a la derecha
275                 current->_right = new optree_node(rpn.pop(), current);
276 
277                 // Si el token era una operación, bajar por derecha
278                 if (is_operation(current->_right->_t))
279                     current = current->_right;
280             }
281             // Derecha ocupada
282             else
283             {
284                 // Izquierda libre
285                 if (current->_left == NULL)
286                 {
287                     // Token a la izquierda
288                     current->_left = new optree_node(rpn.pop(), current);
289 
290                     // Si el token era una operación, bajar por izquierda
291                     if (is_operation(current->_left->_t))
292                         current = current->_left;
```

```
293                 }
294                 // Si ambas ramas están ocupadas, subir
295                 else
296                     current = current->_up;
297             }
298         }
299     }
300 
301     // Asociación del complejo para iterar los pixel
302     this->_asoc_pixel = &z;
303 
304     // Simplificación del árbol, eliminando expresiones 'estáticas'
305     this->_root->simplify();
306 }
307 
308 
309 
310 /*|//////////////////////////////|   4)  |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
311 /*|//////////////////| Operar, para realizar la operación |\\\\\\\\\\\\\\\\\\|*/
312 /*|//////////////////////////////////|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|*/
313 const complex optree::operate() const
314 {
315     // Validaciónes
316     if ( this->_asoc_pixel == NULL || this->_root == NULL ) return complex();
317 
318     return _root->operate(this->_asoc_pixel);
319 }
```

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include "PGMimage.h"
#include "complex.h"
#include "utils.h"
#include "cmdline.h"
#include "node.h"
#include "queue.h"
#include "stack.h"
#include "parser.h"
#include "optree.h"

using namespace std;

////////////////////////////// Variables globales ////////////////////////////
option_t options_[] =           // Opciones CLA
{
    {true,  "i", "input",    "-",       opt_input,    OPT_DEFAULT},
    {true,  "o", "output",   "-",       opt_output,   OPT_DEFAULT},
    {true,  "f", "function", "z",       opt_function, OPT_DEFAULT},
    {true,  "r", "region",   "2,2,0,0", opt_region,   OPT_DEFAULT},
    {false, "h", "help",     NULL,      opt_help,     OPT_DEFAULT},
    {0, },
};
istream      *iss_ = NULL;      // Puntero a stream de entrada
ostream      *oss_ = NULL;      // Puntero a stream de salida
fstream      ifs_;              // Archivo de entrada
fstream      ofs_;              // Archivo de salida
char         *prog_name_;       // Nombre del programa
double       map_w_;            // Ancho de la región de mapeo
double       map_h_;            // Alto de la región de mapeo
complex      map_c_;            // Centro de la región de mapeo
stack<token> rpn_expr_;         // Expresión convertida a RPN


// Macro de función para imprimir mensajes de debugging
#ifdef DEBUG
#define DEBUG_MSG(m) (cerr << m << "\n")
#else
#define DEBUG_MSG(m)
#endif


/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*|||||||||||||||||||||||||||||||||||| Main ||||||||||||||||||||||||||||||||||*/
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
int main(int argc, char** argv)
{
    prog_name_ = argv[0];

    // Validación de argumentos
    cmdline cmdl(options_);
    cmdl.parse(argc, argv);

    DEBUG_MSG("Argumentos validados, apertura de archivos, conversión a RPN.");

    // Lectura del archivo de entrada
    PGMimage in_image;
    if ( !(*iss_ >> in_image) )
    {
        cerr << "Invalid PGM formated input." << endl;
        exit(1);
    }
    // Si nada ha fallado, se puede cerrar el archivo
    if (ifs_) ifs_.close();

    DEBUG_MSG("Archivo de entrada cargado.");

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

```

```
74      PGMimage out_image(w, h, in_image.getColorDepth());
75
76      // Variables para recorrer la imagen
77      complex in_plane, out_plane;
78      size_t i, j, row, col;
79
80      DEBUG_MSG("Imagen de salida y variables auxiliares creadas.");
81
82      // Árbol de la operación
83      optree operation(rpn_expr_, out_plane);
84
85      DEBUG_MSG("Árbol de la operación generado.");
86
87      // Recorrido de la imagen y transformación
88      for (i = 0; i < h; i++)
89      {
90          for (j = 0; j < w; j++)
91          {
92              // Pixel en la imagen de salida <-> Punto en el plano de salida
93              get_complex_from_index(out_plane, i, j, h, w);
94
95              // Aplicación de la operación
96              in_plane = operation.operate();
97
98              // Punto en el plano de entrada <-> Pixel en la imagen de entrada
99              get_row_from_complex(row, in_plane, h);
100             get_col_from_complex(col, in_plane, w);
101
102             // Si no se cayó fuera de la imagen, se copia
103             if (row < h && col < w)
104             {
105                 out_image[i][j] = in_image[row][col];
106             }
107         }
108     }
109
110     DEBUG_MSG("Imagen de salida escrita.");
111
112     // Volcado en el archivo de salida
113     *oss_ << out_image;
114     if (ofs_) ofs_.close();
115
116     DEBUG_MSG("Archivo de salida guardado y cerrado.");
117
118     return 0;
119 }
```

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

# Compilador:
CC = g++
# Flags para linkeo:
LFLAGS = -pedantic -Wall
# Flags para compilación:
CFLAGS = -ansi -pedantic-errors -Wall -O3
# Flags para debugging:
DFLAGS = -g -DDEBUG
# Centinela de debugging:
DEBUG_CENTINEL = .last_debug
DEBUG_CENTINEL_CONTENT = "This file indicates that the last build was made with\
the \'debug\' option."
# Nombre de salida del proyecto:
OUT = tp1
# 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 = $(addprefix $(BIN_DIR)/, \
    PGMimage.o     \
    complex.o      \
    utils.o        \
    cmdline.o      \
    parser.o       \
    optree.o       \
    main.o         \
)
FULL_OUT = $(BIN_DIR)/$(OUT)


# |------------------------| Reglas de construcción |------------------------| #

# Objetivo de fantasía por defecto, para manejar la opción debug
.PHONY: $(OUT)
ifeq (,$(wildcard $(BIN_DIR)/$(DEBUG_CENTINEL)))
# Si no existe el centinela de debug, se procede normalmente
$(OUT): $(FULL_OUT)
else
# Si existe, es necesario limpiar (con lo que también se eliminará el mismo)
$(OUT): clean $(FULL_OUT)
endif

# Construcción del ejecutable de salida
$(FULL_OUT): $(OBJECTS) | $(BIN_DIR)
    $(CC) $(LFLAGS) $(OBJECTS) -o $(FULL_OUT)

# Construcción de los archivos objeto
$(BIN_DIR)/%.o: $(SRC_DIR)/%.cpp
    $(CC) -c $(CFLAGS) $< -o $@

# Creación del directorio de binarios
$(BIN_DIR):
    mkdir $(BIN_DIR)


# |----------------------------| Dependencias |----------------------------| #

```

```
74  $(BIN_DIR)/PGMimage.o: $(addprefix $(SRC_DIR)/, \
75      PGMimage.cpp    \
76      PGMimage.h      \
77  ) | $(BIN_DIR)
78
79  $(BIN_DIR)/complex.o: $(addprefix $(SRC_DIR)/, \
80      complex.cpp     \
81      complex.h       \
82  ) | $(BIN_DIR)
83
84  $(BIN_DIR)/utils.o: $(addprefix $(SRC_DIR)/, \
85      utils.cpp       \
86      utils.h         \
87      complex.h       \
88      cmdline.h       \
89      node.h          \
90      stack.h         \
91      queue.h         \
92      parser.h        \
93  ) | $(BIN_DIR)
94
95  $(BIN_DIR)/cmdline.o: $(addprefix $(SRC_DIR)/, \
96      cmdline.cpp     \
97      cmdline.h       \
98  ) | $(BIN_DIR)
99
100 $(BIN_DIR)/parser.o: $(addprefix $(SRC_DIR)/, \
101     parser.cpp      \
102     parser.h        \
103     node.h          \
104     queue.h         \
105     stack.h         \
106     complex.h       \
107 ) | $(BIN_DIR)
108
109 $(BIN_DIR)/optree.o: $(addprefix $(SRC_DIR)/, \
110     optree.cpp      \
111     optree.h        \
112     complex.h       \
113     node.h          \
114     stack.h         \
115     parser.h        \
116 ) | $(BIN_DIR)
117
118 $(BIN_DIR)/main.o: $(addprefix $(SRC_DIR)/, \
119     main.cpp        \
120     PGMimage.h      \
121     complex.h       \
122     utils.h         \
123     cmdline.h       \
124     node.h          \
125     queue.h         \
126     stack.h         \
127     parser.h        \
128     optree.h        \
129 ) | $(BIN_DIR)
130
131
132
133 # ////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ #
134 # |||||||||||||||||||||||||| Utilidades extras |||||||||||||||||||||||||| #
135 # \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\////////////////////////////////////// #
136
137
138 # |------------------| Debug (compilar con flags de debug) |-----------------| #
139
140 .PHONY: debug
141 debug: CFLAGS += $(DFLAGS)
142 debug: LFLAGS += $(DFLAGS)
143 ifeq (,$(wildcard $(BIN_DIR)/$(DEBUG_CENTINEL)))
144 # Si no existe el centinela de debug, hay que limpiar y crearlo
145 debug: clean $(FULL_OUT)
146     @ echo "$(DEBUG_CENTINEL_CONTENT)" > $(BIN_DIR)/$(DEBUG_CENTINEL)
```

```
147 else
148 # Si existe, solo actualizar si es necesario
149 debug: $(FULL_OUT)
150 endif
151
152
153 # |---------------------------| Limpiar (todo) |---------------------------| #
154
155 .PHONY: clean
156 clean:
157     rm -rf $(BIN_DIR)
158
159
160 # |-----------------------| Limpiar códigos objeto |-----------------------| #
161
162 .PHONY: objclean
163 objclean:
164     rm -f $(BIN_DIR)/*.o
165
166
167 # |---------------| Construir y eliminar archivos temporales |---------------| #
168
169 .PHONY: deltemps
170 deltemps: $(FULL_OUT) objclean
171
172
173 # |-----------------------------| Instalar |-----------------------------| #
174
175 .PHONY: install
176 install: $(FULL_OUT)
177     cp $(FULL_OUT) "$(INSTALL_DIR)"
178
179
180 # |---------------------------| Desinstalar |---------------------------| #
181
182 .PHONY: uninstall
183 uninstall:
184     rm -f "$(INSTALL_DIR)/$(OUT)"
```