Algoritmos y Programación II
Trabajo Práctico 0

Carlos Germán Carreño Romano, Sebastián Sampayo, Rodrigo Bourbon

April 23, 2015

Contents

1 Enunciado **1**

2 Introducción **1**
2.1 Radio definida por software (SDR) 1
2.2 Transmisión de TV por cable 2
2.3 Aplicación del Trabajo Práctico 2

3 Estándar de estilo **3**

4 Estructura de archivos **3**

5 Opciones de ejecución **3**

6 Compilación **3**

7 Casos de prueba **3**

8 Desarrollo **3**

9 Códigos **3**

1 Enunciado

2 Introducción

2.1 Radio definida por software (SDR)

El concepto de Radio Definida por Software se le atribuye a Joseph Mitola, 1990. Se refiere a un dispositivo que permite reducir al mínimo el hardware necesario para la recepción de señales de radio. Dicho equipo captura la señal analógica (ya sea mediante un cable o una antena), la digitaliza (mediante un conversor A/D) para luego realizar por software toda la etapa de procesamiento de señal requerido en la decodificación. Esto ha logrado que la recepción de cierto rango de telecomunicaciones sea mucho más accesible en términos económicos y prácticos (ya que el mismo dispositivo físico se puede utilizar para distintos fines

Figure 1: Sintonizador de radio digital.

con solo re-programar el software). Un ejemplo de este dispositivo se puede ver a continuación:

2.2 Transmisión de TV por cable

En telecomunicaciones, la televisión analógica se transmite mediante el método de la Multiplexión por División en Frecuencia (FDM). Esta técnica consiste en transmitir varias señales simultáneamente modulando cada una con una portadora diferente, en el rango de VHF/UHF, de forma tal que los anchos de banda de cada señal no se superpongan significativamente. El canal destinado para la transmisión de una emisora tiene un ancho de banda de aproximadamente 6 Mhz, donde los 5.45 Mhz más bajos corresponden al espectro de la señal de video y los últimos 0.55Mhz (aproximadamente) se reservan para el espectro de la señal de audio. Este modelo de comunicación se puede ver representado en el siguiente gráfico:

2.3 Aplicación del Trabajo Práctico

Sabiendo que el audio de la televisión está modulado en frecuencia (FM), si se toma la porción del canal adecuada es posible demodular dicha señal y escuchar algún canal de televisión.

En este caso particular, el SDR se utilizó para capturar un ancho de banda de 2.4Mhz y centrado en 181.238 Mhz. A través del aplicativo desarrollado se pudo escuchar efectivamente el programa emitido.

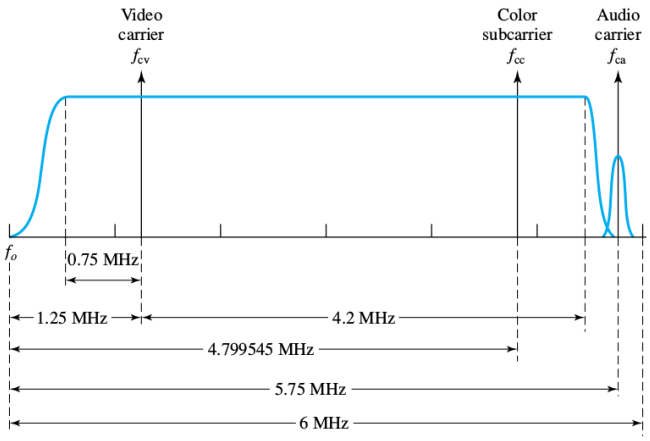

Figure 2: Señal de TV transmitida.

3 Estándar de estilo

Adoptamos la convención de code styling de Google para C++, salvando las siguientes excepciones:

- streams: utilizamos flujos de entrada y salida
- sobrecarga de operadores
-

4 Estructura de archivos

5 Opciones de ejecución

6 Compilación

7 Casos de prueba

8 Desarrollo

9 Códigos

main.cc

```
1 //
                                                                    //
```

```
2  // Facultad de Ingenieria de la Universidad de Buenos
       Aires
3  // Algoritmos y Programacion II
4  // 1er Cuatrimestre de 2015
5  // Trabajo Practico 0: Programacion en C++
6  // Demodulacion de senal FM
7  //
8  // main.cc
9  // Archivo principal donde se ejecuta el main.
10 //
       ------------------------------------------------------------------------
       //
11
12 #include <iostream>
13 #include <fstream>
14 #include <sstream>
15 #include <cstdlib>
16 #include <cstring>
17
18 #include "main.h"
19 #include "complex.h"
20 #include "cmdline.h"
21 #include "arguments.h"
22 #include "utilities.h"
23 #include "types.h"
24
25 using namespace std;
26
27
28 // Coleccion de funciones para imprimir en formatos
       distintos
29 void (*print_phase[])(double) = {
30
31   print_phase_text,
32   print_phase_U8
33
34 };
35
36 // Opciones de argumentos de invocacion
37 static option_t options[] = {
38   {1, "i", "input", "-", opt_input, OPT_SEEN},
39   {1, "o", "output", "-", opt_output, OPT_SEEN},
40   {1, "f", "format", "txt", opt_format, OPT_SEEN},
41   {0, },
42 };
43
44 extern istream *iss;
45 extern format_option_t format_option;
46
47 int
```

```
48 main(int argc, char * const argv[])
49 {
50   size_t i = 0;
51   size_t j = 0;
52   Complex buffer1 = 0;
53   double buffer2 = 0;
54   Complex x, x_prev;
55   Complex aux;
56   double output_phase;
57
58   // Parsear argumentos de invocacion
59   cmdline cmdl(options);
60   cmdl.parse(argc, argv);
61
62   // Inicializar el complejo con el fomato especificado
63   Complex input_complex(format_option);
64   // cout<<int(input_complex.format_option())<<endl;
65
66   // ( -- Condiciones Iniciales Nulas -- )
67   x_prev = 0;
68
69   // Mientras haya complejos en la entrada
70   while (*iss >> input_complex)
71   {
72     // ( -- Promediar 11 elementos -- )
73     buffer1 += input_complex;
74     if (i < DECIMATOR1_SIZE-1)
75     {
76       i++;
77       continue;
78     }
79     x = buffer1/DECIMATOR1_SIZE;
80     buffer1 = 0;
81     i = 0;
82
83     // ( -- Obtener la derivada de la fase -- )
84     aux = x * x_prev.conjugated();
85     output_phase = aux.phase();
86
87     // ( -- Avanzar una muestra -- )
88     x_prev = x;
89
90     // ( -- Promediar 4 elementos -- )
91     buffer2 += output_phase;
92     if (j < DECIMATOR2_SIZE-1)
93     {
94       j++;
95       continue;
96     }
97     output_phase = buffer2/DECIMATOR2_SIZE;
```

```
98        buffer2 = 0;
99        j = 0;
100
101       // ( -- Imprimir en el formato especificado -- )
102       (print_phase[format_option])(output_phase);
103     }
104     return 0;
105   } // main
```

complex.cc

```
1   #include <iostream>
2   #include <cmath>
3
4   #include "complex.h"
5   #include "types.h"
6
7   using namespace std;
8
9
10
11  Complex::Complex() : real_(0), imag_(0), format_option_(
        FORMAT_OPTION_TEXT)
12  {
13  }
14
15  Complex::Complex(format_option_t f_o) : real_(0), imag_
        (0), format_option_(f_o)
16  {
17  }
18
19  Complex::Complex(double r) : real_(r), imag_(0),
        format_option_(FORMAT_OPTION_TEXT)
20  {
21  }
22
23  Complex::Complex(double r, double i) : real_(r), imag_(i)
        , format_option_(FORMAT_OPTION_TEXT)
24  {
25  }
26
27  Complex::Complex(double r, double i, format_option_t f_o)
        : real_(r), imag_(i), format_option_(f_o)
28  {
29  }
30
31  Complex::Complex(Complex const &c) : real_(c.real_),
        imag_(c.imag_), format_option_(c.format_option_)
32  {
33  }
```

```
34
35  Complex const &
36  Complex::operator=(Complex const &c)
37  {
38    real_ = c.real_;
39    imag_ = c.imag_;
40    format_option_ = c.format_option_;
41    return *this;
42  }
43
44  Complex const &
45  Complex::operator*=(Complex const &c)
46  {
47    double re = real_ * c.real_
48              - imag_ * c.imag_;
49    double im = real_ * c.imag_
50              + imag_ * c.real_;
51    real_ = re, imag_ = im;
52    return *this;
53  }
54
55  Complex const &
56  Complex::operator+=(Complex const &c)
57  {
58    double re = real_ + c.real_;
59    double im = imag_ + c.imag_;
60    real_ = re, imag_ = im;
61    return *this;
62  }
63
64  Complex const &
65  Complex::operator-=(Complex const &c)
66  {
67    double re = real_ - c.real_;
68    double im = imag_ - c.imag_;
69    real_ = re, imag_ = im;
70    return *this;
71  }
72
73  Complex::~Complex()
74  {
75  }
76
77  double
78  Complex::real() const
79  {
80    return real_;
81  }
82
83  double Complex::imag() const
```

```
84  {
85    return imag_;
86  }
87
88  format_option_t Complex::format_option() const
89  {
90
91    return format_option_;
92
93  }
94
95  double
96  Complex::abs() const
97  {
98    return std::sqrt(real_ * real_ + imag_ * imag_);
99  }
100
101 double
102 Complex::abs2() const
103 {
104   return real_ * real_ + imag_ * imag_;
105 }
106
107 double
108 Complex::phase() const
109 {
110   return atan2(imag_, real_);
111 }
112
113 Complex const &
114 Complex::conjugate()
115 {
116   imag_*= -1;
117   return *this;
118 }
119
120 Complex const
121 Complex::conjugated() const
122 {
123   return Complex(real_, -imag_);
124 }
125
126 bool
127 Complex::iszero() const
128 {
129 #define ZERO(x) ((x) == +0.0 && (x) == -0.0)
130   return ZERO(real_) && ZERO(imag_) ? true : false;
131 }
132
133 Complex const
```

```
134 operator+(Complex const &x, Complex const &y)
135 {
136   Complex z(x.real_ + y.real_, x.imag_ + y.imag_);
137   return z;
138 }
139
140 Complex const
141 operator-(Complex const &x, Complex const &y)
142 {
143   Complex r(x.real_ - y.real_, x.imag_ - y.imag_);
144   return r;
145 }
146
147 Complex const
148 operator*(Complex const &x, Complex const &y)
149 {
150   Complex r(x.real_ * y.real_ - x.imag_ * y.imag_,
151             x.real_ * y.imag_ + x.imag_ * y.real_);
152   return r;
153 }
154
155 Complex const
156 operator/(Complex const &x, Complex const &y)
157 {
158   return x * y.conjugated() / y.abs2();
159 }
160
161 Complex const
162 operator/(Complex const &c, double f)
163 {
164   return Complex(c.real_ / f, c.imag_ / f);
165 }
166
167 bool
168 operator==(Complex const &c, double f)
169 {
170   bool b = (c.imag_ != 0 || c.real_ != f) ? false : true;
171   return b;
172 }
173
174 bool
175 operator==(Complex const &x, Complex const &y)
176 {
177   bool b = (x.real_ != y.real_ || x.imag_ != y.imag_) ?
       false : true;
178   return b;
179 }
180
181 istream &
182 operator>>(istream &is, Complex &c)
```

```
183 {
184   switch (c.format_option_) {
185
186     case FORMAT_OPTION_TEXT:
187       return read_format_text(is, c);
188
189     case FORMAT_OPTION_U8:
190       return read_format_U8(is, c);
191
192   }
193
194   return is;
195 }
196
197 ostream &
198 operator<<(ostream &os, const Complex &c)
199 {
200   switch (c.format_option_) {
201
202     case FORMAT_OPTION_TEXT:
203       return write_format_text(os, c);
204
205     case FORMAT_OPTION_U8:
206       return write_format_U8(os, c);
207
208   }
209
210   return os;
211 }
212
213 // Lee en formato texto "(Real, Imaginario)"
214 istream &
215 read_format_text(istream &is, Complex &c)
216 {
217
218   int good = false;
219   int bad  = false;
220   double re = 0;
221   double im = 0;
222   char ch = 0;
223
224   if (is >> ch
225       && ch == '(') {
226       if (is >> re
227         && is >> ch
228         && ch == ','
229         && is >> im
230         && is >> ch
231         && ch == ')')
232         good = true;
```

```
233       else
234           bad = true;
235     } else if (is.good()) {
236         is.putback(ch);
237       if (is >> re)
238         good = true;
239       else
240       bad = true;
241     }
242
243     if (good)
244       c.real_ = re, c.imag_ = im;
245     if (bad)
246       is.clear(ios::badbit);
247
248     return is;
249
250   }
251
252   // Lee en formato binario Real8bits seguido de
        Imaginario8bits
253   istream &
254   read_format_U8(istream &is, Complex &c)
255   {
256
257     int good = false;
258     int bad  = false;
259     unsigned char re = 0;
260     unsigned char im = 0;
261
262     if (is >> re && is >> im)
263     good = true;
264     else
265       bad = true;
266
267     if (good)
268     c.real_ = re - 128;
269     c.imag_ = im - 128;
270     if (bad)
271     is.clear(ios::badbit);
272
273     return is;
274
275   }
276
277   // Escribe en formato texto "(Real, Imaginario)"
278   ostream &
279   write_format_text(ostream &os, const Complex &c)
280   {
281
```

```
282   return os << "("
283             << c.real()
284             << ", "
285             << c.imag()
286             << ")";
287
288 }
289
290 // Escribe en fomato binario Real8bits seguido de
      Imaginario8bits
291 ostream &
292 write_format_U8(ostream &os, const Complex &c)
293 {
294
295   return os << (char)c.real()
296             << (char)c.imag();
297
298 }
```

arguments.cc

```
1  //
       ------------------------------------------------------------------------
       //
2  // Facultad de Ingenieria de la Universidad de Buenos
       Aires
3  // Algoritmos y Programacion II
4  // 1er Cuatrimestre de 2015
5  // Trabajo Practico 0: Programacion en C++
6  // Demodulacion de senal FM
7  //
8  // arguments.cc
9  // Funciones a llamar para cada opcion posible de la
       aplicacion
10 //
       ------------------------------------------------------------------------
       //
11
12 #include <iostream>
13 #include <fstream>
14 #include <sstream>
15 #include <cstdlib>
16 #include <cstring>
17
18 #include "arguments.h"
19 #include "types.h"
20
21 using namespace std;
22
23
```

```
24
25 istream *iss;
26 ostream *oss;
27 fstream ifs;
28 fstream ofs;
29 format_option_t format_option;
30
31 // Nombres de los argumentos de la opcion "--format"
32 string description_format_option[] = {
33
34   FORMAT_TEXT,
35   FORMAT_U8
36
37 };
38
39 void
40 opt_input(string const &arg)
41 {
42   // Si el nombre del archivos es "-", usaremos la
       entrada
43   // estandar. De lo contrario, abrimos un archivo en
       modo
44   // de lectura.
45   //
46   if (arg == "-") {
47     iss = &cin;     // Establezco la entrada estandar cin
         como flujo de entrada
48   }
49   else {
50     ifs.open(arg.c_str(), ios::in); // c_str(): Returns a
         pointer to an array that contains a null-
         terminated
51                     // sequence of characters (i.e., a C-
                           string) representing
52                     // the current value of the string
                           object.
53     iss = &ifs;
54   }
55
56   // Verificamos que el stream este OK.
57   //
58   if (!iss->good()) {
59     cerr << "Cannot open "
60          << arg
61          << "."
62          << endl;
63     exit(1);
64   }
65 }
66
```

```
67  void
68  opt_output(string const &arg)
69  {
70    // Si el nombre del archivos es "-", usaremos la salida
71    // estandar. De lo contrario, abrimos un archivo en
        modo
72    // de escritura.
73    //
74    if (arg == "-") {
75      oss = &cout;  // Establezco la salida estandar cout
          como flujo de salida
76    } else {
77      ofs.open(arg.c_str(), ios::out);
78      oss = &ofs;
79    }
80
81    // Verificamos que el stream este OK.
82    //
83    if (!oss->good()) {
84      cerr << "Cannot open "
85           << arg
86           << "."
87           << endl;
88      exit(1);    // EXIT: Terminacion del programa en su
          totalidad
89    }
90  }
91
92  void
93  opt_format(string const &arg)
94  {
95    size_t i;
96    // Recorremos diccionario de argumentos hasta encontrar
        uno que coincida
97    for(i=0; i < FORMAT_OPTIONS; i++) {
98      if(arg == description_format_option[i]) {
99        format_option = (format_option_t)i;  // Casteo
100       break;
101     }
102   }
103   // Si recorrio todo el diccionario, el argumento no
        esta implementado
104   if (i == FORMAT_OPTIONS) {
105     cerr << "Unknown format" << endl;
106     exit(1);
107   }
108 }
```