

# 75.04/95.12 Algoritmos y Programación II

## Trabajo práctico 0: Programación C++

Universidad de Buenos Aires - FIUBA  
Primer cuatrimestre de 2015  
\$Date: 2015/04/06 01:03:33 \$

### 1. Objetivos

Ejercitar conceptos básicos de programación C++, implementando un programa y su correspondiente documentación que resuelva el problema descripto más abajo.

### 2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

### 3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

### 4. Introducción

En este trabajo implementaremos una herramienta para procesar señales de audio moduladas en frecuencia de acuerdo con el flujo de cómputo que figura a continuación.

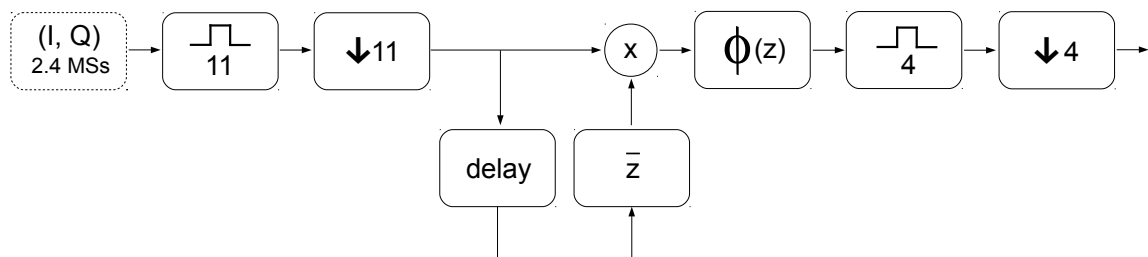


Figura 1: diagrama de flujo de procesamiento de señales de RF

El programa recibirá un *stream* de números complejos  $(I, Q)$  que digitalizan una porción de 2.4 MHz del espectro de RF, centrado en la frecuencia de la estación de radio que queremos demodular. Estos números ingresarán al flujo de procesamiento de señales de la figura 1, en donde se realizan tres funciones esenciales:

- **Filtrado del espectro útil de la banda digitalizada.** Los primeros 2 bloques del demodulador se encargan de extraer la banda de 220 kHz donde reside la información útil, y eliminar las interferencias que podrían encontrarse en otras frecuencias dentro de la porción de espectro digitalizada.
- **Extracción de la señal de audio.** La señal filtrada y decimada ingresa a los bloques centrales en donde se extraen las variaciones de fase instantáneas generando un *stream* de salida constituido por números reales entre  $-\pi$  y  $\pi$ .
- **Ajuste de velocidad de datos y reescalamiento.** La etapa final busca preparar la señal para ser enviada al sistema de sonido: luego de reducir la frecuencia de muestreo 4 veces, el *stream* es escalado de forma tal que las variaciones de fase entre  $-2\pi$  y  $2\pi$  se trasladen al intervalo de valores comprendidos entre  $+1$  y  $-1$ , en forma lineal. Así, el *stream* de salida queda listo para ser enviado al sistema de sonido.

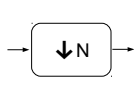
## 4.1. Componentes

A continuación se detallan los bloques funcionales del flujo de procesamiento de señales presentado en la introducción.



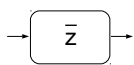
$y[n] = \sum_{k=0}^{N-1} x[n-k]$

**Moving average.** Promedia las últimas  $N$  muestras recibidas en cada instante de tiempo.



$y[n] = x[n \cdot N]$

**Decimador.** Reduce la velocidad de muestreo de la señal de entrada, seleccionando una de cada  $N$  muestras adyacentes.



$y[n] = \overline{x[n]}$

**Conjugado.** Calcula el valor conjugado complejo de la señal de entrada.



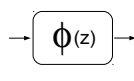
$y[n] = x[n - 1]$

**Delay.** Retrasa la señal de entrada en un instante de tiempo.



$y[n] = x_1[n] \cdot x_2[n]$

**Producto.** Multiplica el valor de ambas señales.



$y[n] = \Phi(x[n])$

**Fase.** Calcula la fase de la señal de entrada compleja, retornando un valor comprendido entre  $-\pi$  y  $\pi$ .

## 5. Programa

### 5.1. Interfaz

Tanto en este TP, como en el siguiente, la interacción con el programa se dará a través de la línea de comando teniendo en cuenta las características descriptas a continuación.

**Entrada.** Los datos de entrada ingresan a nuestro programa en forma de números complejos de la forma  $(I, Q)$ , en donde  $I$  representa la componente en fase y  $Q$  el valor en cuadratura.

A lo largo de este TP, supondremos que la entrada/salida es realizada en formato texto: es decir, la entrada está conformada por una secuencia de complejos separados por espacios, en donde cada complejo se representa en forma de par ordenado.

**Salida.** El *stream* de salida producido por este programa estará conformado por valores escalares separados por espacios, en donde cada valor está comprendido entre +1 y -1.

**Línea de comando.** Las opciones `-i` y `-o` permitirán seleccionar los *streams* de entrada y salida de datos respectivamente. Por defecto, éstos serán `cin` y `cout`. Lo mismo ocurrirá al recibir “-” como argumento de cada una.

Al finalizar, todos nuestros programas retornarán un valor nulo en caso de no detectar ningún problema; y, en caso contrario, devolveremos un valor no nulo (por ejemplo 1).

Como comentario adicional, el orden de las opciones es irrelevante. Por este motivo, no debe asumirse un orden particular de las mismas a la hora de desarrollar la toma de argumentos.

## 5.2. Optativo

Como explicamos en la sección anterior, el comportamiento por defecto del programa es procesar información en formato texto. Se propone entonces introducir una opción, `-f` o `--format`, que permita parametrizar este comportamiento:

- Cuando el programa recibe `txt` como argumento de `-f`, el programa realizará el comportamiento descrito en la sección anterior (procesa archivos de texto con formato). Lo mismo ocurrirá cuando esta opción no esté presente en la línea de comando.
- En cambio, cuando es invocado con `-f U8`, permitirá procesar las muestras generadas por el sistema de radio: magnitudes con precisión de 8 bits sin signo, en forma de un *stream* crudo que alterna los valores de byte de `I` y `Q` comprendidos entre 0 y 255.

Tener en cuenta que la entrada se encuentra codificada en formato `U8`: es decir, está constituida por escalares de 8 bits sin signo, siendo el valor de reposo de la señal igual a 128. Así, tenemos por ejemplo que el par de bytes (128, 128) se corresponde con el complejo (0, 0); mientras que (130, 127) representa al valor (2, -1), etc.

Se incluyen un par de capturas de referencia codificadas en este formato para aquellos que opten por implementar esta funcionalidad [1] [2].

## 5.3. Ejemplos

**Caso 1.** Comenzamos transformando un *stream* de muestras vacío:

```
$ touch /tmp/1.in
$ ls -l /tmp/1.in
-rw-rw-r-- 1 user group 0 Apr  4 14:59 /tmp/1.in
$ ./tp0 -i /tmp/1.in -o /tmp/1.out
$ ls -l /tmp/1.out
-rw-rw-r-- 1 user group 0 Apr  4 15:00 /tmp/1.out
```

**Caso 2.** A continuación, generemos secuencia de 44 complejos con valor nulo. De acuerdo con lo discutido en la introducción, la salida deberá estar compuesta por un único valor:

```
$ perl -e 'print "(0, 0)\n" x 44;' >/tmp/2.in
$ ./tp0 -i /tmp/2.in -o -
0
```

**Caso 3.** En este ejemplo, usamos el programa `test3.pl` para generar una secuencia de complejos que vayan rotando la fase a un ritmo constante de  $\pi/10$ .

```
$ cat test3.pl
#!/usr/bin/perl
#
# test3.pl - genera una secuencia de pares complejos rotando en
# incrementos de pi/10, generando un valor constante en la salida
# demodulada.
#
# \Id: tp0-9512-2015-1q.tex.in,v 1.10 2015/04/06 01:03:33 lesanti Exp \Id:
$pi = abs(atan2(0, -1));

for (1 .. 4) {
    for (1 .. 4) {
        $phi += $pi / 10;
        $re = cos($phi);
        $im = sin($phi);
        $sign *= -1.0;

        print "($re, $im)\n" x 11;
    }
}
```

Al pasar estos datos por el TP, la salida es:

```
$ perl ./test3.pl | ./tp0 -i - -o -
0.0725588
0.0999948
0.1
0.1
```

Notar que la salida es exactamente 0.1, salvo en el primer caso el cual se ve afectado por las condiciones iniciales (nulas) del flujo de procesamiento de señales.

**Caso 4.** Generamos una secuencia de números complejos alternando el signo de la fase, reflejándose este comportamiento en la salida demodulada:

```
$ perl ./test4.pl | ./tp0 -i - -o - | head
0
-0.0249983
0.0249966
-0.0249966
0.0249966
-0.0249966
0.0249966
-0.0249966
0.0249966
-0.0249966
```

**Caso 5 (optativo).** Decodificamos el archivo [2], enviando la salida al sistema de sonido (este archivo contiene 60s de captura del espectro de Tv por cable sintonizado en 181.238 MHz, muestreando a una velocidad de 2.400.000 muestras (I, Q) por segundo:

```
$ ./tp0 -f U8 -i sample-2-2.4Mss -o - | aplay - -f U8 -r 54000
...
```

Se incluye también un archivo equivalente, `sample-1-1.0Mss` [1], muestreado a menor velocidad para casos en los cuales el sistema no tenga suficiente poder de cómputo para reproducir el contenido en tiempo real.

```
$ ./tp0 -f U8 -i sample-1-1.0Mss -o - | aplay - -f U8 -r 22700
...
```

Como siempre, estos ejemplos deben ser incluidos como punto de partida de los casos de prueba del trabajo práctico.

## 5.4. Portabilidad

Es deseable que la implementación desarrollada provea un grado mínimo de portabilidad. Sugerimos verificar nuestros programas en alguna versión reciente de UNIX: BSD o Linux.

## 6. Informe

El contenido mínimo del informe deberá incluir:

- Una carátula que incluya los nombres de los integrantes y el listado de todas las entregas realizadas hasta ese momento, con sus respectivas fechas.
- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C++ (en dos formatos, digital e impreso).
- Este enunciado.

## 7. Fechas

La última fecha de entrega y presentación será el jueves 23/4.

## Referencias

- [1] <http://fiuba7504.com.ar/sample-1-1.0Mss.zip>
- [2] <http://fiuba7504.com.ar/sample-2-2.4Mss.zip>
- [3] Código fuente para correr los casos de prueba 3 y 4. <http://materias.fi.uba.ar/7504E/material/test.zip>