Aleatoriedad

Random

Esp. Ing. José María Sola, profesor.

Revisión 1.0.0
2018-07-04

Tabla de contenidos

1

Introducción

La aleatoriedad tiene varias aplicaciones en la informática:

- Simulaciones para la toma de decisiones.
- Generación de muestreos estadísticos.
- Criptografía.
- Implementación del componente azaroso en juegos.
- Implementación de asignaciones no determinísticas.

Con respecto a C, C++ presenta un modelo para implementar aleatoriedad mucho más flexible y eficiente que las funciones `srand` y `rand` de C. Es más flexible porque:

- Permite optar que la semilla de la secuencia aleatoria la provea el hardware, si el sistema lo permite, es decir, no dependemos de un función que retorne el instante actual de tiempo (`time`).
- Permite seleccionar entre varios motores predefinidos que implementan algoritmos generadores de números aleatorios, o inclusive nos permite definir nuestro propio generador.
- Incluye distribuciones predefinidas que transforman la salida del generador según la función de probabilidad que requiramo, y también permite definir nuestras propias distribuciones.

Por otro lado, el modelo es más eficaz comparado con el de C, ya que es común que las implementaciones de la función `rand` tengan un bug que otorga más frecuencia a algunos números por sobre otros, el problema se evidencia

sobre los bits de menor orden, los cuales utiliza al realizar la operación resto (`%`). [MANRAND] [CPPRRAND].

Como contrapartida, el uso de este modelo flexible resulta más complejo comparado al par de funciones `srand` y `rand`.

Este texto presenta el modelo que propone C++ con ejemplos simples. La eficacia y flexibilidad sobrepasan la complejidad relativa de su uso.

2

Pasos Generales para usar para Generar Números Aleatorios

1. Preparación. Antes de obtener un número aleatorio necesitamos realizar una preparación previa:
    a. Declarar un dispositivo aleatorio que vamos a usar como semilla que inicializa la secuencia de números aleatorios.
    b. Declarar una generador (i.e., secuenciador) de números aleatorios, incializado con el dispositivo anterior.
    c. Declarar una distribución para transformar el número aleatorio según alguna función de probabilidad.
2. Obtención del número aleatorio.
    a. Solicitar a la distribución que genere un número aleatorio según su función de probabilidad y dado el generador anterior.

3

Ejemplo Introductorio

El ejemplo muestra un histograma para verificar si los número son realmente aleatorios y con una distribución uniforme. En el ejemplo usamos:

- El dispositivo aleatorio del sistema, asumimos está presente.

Lamentablemente el popular compilador mingw[1] tiene un bug asociado a este punto, generado por una función de la API de Windows.

Para el detalle ver: https://sourceforge.net/p/mingw-w64/bugs/338/

Para una solución: https://github.com/euloanty/mingw-std-random_device

- El generador de números aleatarios por omisión.
- Una distribución informe en el rango de enteros [0,7].

```
/* Random
JMS
2018-06-03*/

#include <random> // random_device default_random_engine
 uniform_int_distribution
#include <iostream>
#include <array>
```

[1] https://mingw-w64.org

```
int main(){
 std::random_device r; ❶
 std::default_random_engine e(r()); ❷
 std::uniform_int_distribution<int> du(0, 7); ❸

 std::array<int,8> h{0}; ❹
 for(int i=0; i<1000000; ++i)
  ++h.at(du(e)); ❺

 for(int i=0; i<8; ++i) ❻
  std::cout << i << ": " << h.at(i) << '\n';
}
```

❶ Proveedor de número aleatorio, el primero se usa como semilla del secuenciador.

❷ Secunciador de números aleatorio por defecto, inicializado con la semilla.

❸ Distribuidor uniforme

❹ Genera el histograma

❺ Cuenta el ítem en `du(e)`. `du(e)` obtiene el siguiente número aleatorio distribuido uniformemente entre 0 y 7.

❻ Muestra el histograma

4

Ejemplo con Retiro Aleatorio sin Reposición: ¡Bingo!

Este ejemplo permite implementar el típico caso del sorteo de los bolilleros que se usan en un Bingo.

```
/* Bolilero
JMS
2018-06-03*/

#include <iostream> // cout
#include <string>   // string
#include <iomanip>  // setw
#include <random>   // random_device default_random_engine
 uniform_int_distribution

int main(){
 std::random_device r;
 std::default_random_engine e(r());
 std::string bolillas{"ABCDEFGH"};
 auto sorteos{bolillas.length()}; // Cantidad de sorteos
 char SacarBolillaAleatoria(std::string&, std::default_random_engine&);

 for(unsigned n=0; n < sorteos; ++n){
  std::cout
   << "Sorteo: "      << n
   << "\tBolillero: " << std::setw(sorteos) << bolillas
   << "\tBolilla: "   << SacarBolillaAleatoria(bolillas, e) << "\n";
 }
}
```

```
char SacarBolillaAleatoria(std::string& bolillas,
 std::default_random_engine& e){
 char Sacar(std::string&, unsigned);
 std::uniform_int_distribution<int> du(0, bolillas.length()-1); //
 Distribuidor uniforme
 return Sacar(bolillas, du(e)); // Retorna el caracter que saca de la
 posición aleatoria du(e).
}

char Sacar(std::string& s, unsigned i){
 char c{s.at(i)}; // Copia el elemento que está en esa posición.
 s.erase(i, 1);   // Remueve el elemento de esa posición.
 return c;        // Retorna la copia.
}
```

Bibliografía

http://man7.org/linux/man-pages/man3/rand.3.html#NOTES

https://en.cppreference.com/w/c/numeric/random/rand#Notes