GENERAR CASOS DE PRUEBA

next\_permutation() 🡪 Genera cíclicamente la siguiente permutación de una secuencia en orden lexicográfico. Devuelve falso si ha producido la permutación ordenada y verdadero en caso contrario.

Se usa empezando por una permutación oredenada, de esta manera generará todas las permutaciones posibles hasta llegar de nuevo al vector ordenado (Que en ese caso devuelve falso).

EJEMPLO:

|  |
| --- |
|  |
| const unsigned N = 8;  int v[N], // Permutación probada  w[N], // Resultado esperado  i = 0; // Contador de pruebas  for (int n = 1; n <= N; ++n) {  v[n − 1] = n − 1; // Permutación actual ordenada  memcpy(w, v, sizeof v); // Vector ordenado  do {  if (!prueba\_ordena(v, w, n)){  cout << "ERROR." << endl;  return 1;  }  }while (next\_permutation(v, v + n)); // Siguiente permutación de v  } |
| next\_permutation(v, v + n) genera la siguiente permutación de v y la guarda en v, cuando el vector vuelve llegar a la permutación ordenada del vector deveulve false y se para el do while, luego el for incrementa el tamaño del vector v y se vuelve a probar todas las permutaciones en el do while |

GENERAR NÚMEROS PSEUDOALEATORIOS

rand 🡪 Devuelve un entero aleatorio entre 0 y RAND\_MAX

sRand(int ) 🡪 Asigna un valor a la semilla que será usada por rand para generar números pseudoaleatorios (si no se asigna se usa el 1).

Con la misma semilla rand() generará los mismos números aleatorios, es decir, si le asignamos un valor de 3 a la semilla, ejecutamos el programa en el que rellena un vector usando rand() en un bucle con por ejemplo estos número [23412, 942, 7854]. Si volvemos a ejecutar el programa nos rellenará el vector con exactamente los mismos números.

Por lo que si queremos que genere otros números pseudoaleatorios debemos asignar a la semilla un valor que varie con cada ejecución como el tiempo (usamos la función time() de la librería ctime).

time() 🡪Devuelve un tipo std::time\_t (entero) que representa los segundos que han pasado desde el 1/1/1970 hasta ahora y con lo que se obtiene la hora actual.

EJEMPLO:

|  |
| --- |
|  |
| int main()  srand(time(nullptr)); // Asignamos un valor a la semilla  for (int i = 0; i < 10; ++i)  cout << rand() << endl;  } |
| En este programa generamos 10 números pseudoaleatorios y como asignamos una semilla diferente en cada ejecución gracias a time(), cada vez que se ejecute serán otros números aleatorios diferentes. |

PERMUTACIONES PSEUDOALEATORIAS

random\_shuffle() 🡪Genera una permutación aleatoria del vector que le pasemos como parámetro.

Al igual que rand() es una generación pseudoaleatoria por lo que con la misma semilla y mismo vector nos genera el mismo vector con cada ejecución, en este caso no podemos cambiar la semilla por lo que si queremos cambiar la generación tendremos que cambiar el vector.

|  |
| --- |
|  |
| int v[] = { 0, 1, 2, 3, 4, 5, 6, 7 };  const size\_t n = sizeof v / sizeof ∗v;  random\_shuffle(v, v + n); |
|  |

**EJERCICIO 1**

Utilizando rand(), escriba una función que genere un número pseudoaleatorio entero en el intervalo discreto [a, b].

* a = limite inferior
* b = limite superior

int numAleatorio(int a, int b){

return a+rand()%(b+1-a);

}

**EJECICIO 2**

Escriba un programa que simule 10 000 000 de tiradas de un dado. Imprima las frecuencias relativas que se obtienen para cada una de las caras. Emplee la función del ejercicio anterior.

int main(){

int n=N, i, num;

float d1=0, d2=0, d3=0, d4=0, d5=0, d6=0;

for(i=0 ; i<n ; i++){

switch (numAleatorio(1,6)){

case 1: d1++; break;

case 2: d2++; break;

case 3: d3++; break;

case 4: d4++; break;

case 5: d5++; break;

case 6: d6++; break;

}

}

cout << "(1) F: " << d1 << endl;

cout << "(2) F: " << d2 << endl;

cout << "(3) F: " << d3 << endl;

cout << "(4) F: " << d4 << endl;

cout << "(5) F: " << d5 << endl;

cout << "(6) F: " << d6 << endl;

}

**EJERCICIO 3**

Utilizando rand(), escriba una función que genere un número pseudoaleatorio de coma flotante y precisión doble en el intervalo continuo [a, b].