En cada ejercicio se evaluará la eficiencia del código, el uso de identificadores significativos, la indentación, escritura correcta de llaves {} y el uso adecuado de la palabra reservada const. Se dispone de dos horas para entregar la prueba y debe realizarse en forma estrictamente individual.

Un *cuadrado latino* es una matriz cuadrada, de `n x n` elementos, donde un elemento sólo puede aparecer una vez en cada fila y una única vez en cada columna. Los siguientes son dos ejemplos de cuadrados latinos:

1	2	3
2	3	1
3	1	2

camello	burro	dromedario	asno
burro	asno	camello	dromedario
asno	dromedario	burro	camello
dromedario	camello	asno	burro

Los cuadrados latinos tienen varias aplicaciones, como aleatorizar tratamientos en un experimento o juegos como Sudoku. En esta oportunidad se necesita una clase `LatinSquare` la cual construye cuadrados latinos de tamaño arbitrario de elementos de cualquier tipo de datos. Un ejemplo de uso de la clase se encuentra al final de este enunciado. Un cuadrado latino se puede construir a partir de otro intercambiando dos filas o dos columnas cualquiera. Su clase `LatinSquare` debe ser capaz de generar todos los posibles cuadrados latinos para un tamaño dado. La clase `LatinSquare` debe implementar lo siguiente:

1. [10%] **Miembros de datos**. Debe almacenar la matriz de elementos de cualquier tipo de datos en memoria dinámica. Tenga en cuenta que la expresión `new DataType[n][m]`, donde `DataType` es un tipo de datos cualquiera, no es válida en C++.
2. [15%] **Constructor**. Recibe un vector de C (y el tamaño de éste) con los símbolos que deben usarse para construir el cuadrado latino. Debe construir un cuadrado latino inicial válido.
3. [10%] **Operador <<** imprime el cuadrado latino a un archivo o la salida estándar. Separan los elementos por un espacio en blanco (o si prefiere, un tabulador) y las filas por cambios de línea.
4. [15%] **Operador ++** (preincremento). Transforma un cuadrado latino en otro cuadrado latino distinto y válido (no crea un nuevo objeto, sino que modifica el existente). Este método debe delegar responsabilidad en otros como `swapRows()` o `swapColumns()`.
5. [10%] **Método `swapRows()` y `swapColumns()`**. Intercambia dos filas distintas de la matriz, o dos columnas distintas de la matriz, respectivamente.
6. [10%] **Operador de conversión a booleano**. Un objeto `LatinSquare` está diseñado para transformarse en todos los posibles cuadrados latinos para un tamaño dado. El operador ++ cambia de un cuadrado latino válido a otro. Cuando un objeto `LatinSquare` ya se ha transformado en todas las posibilidades, se convierte en un cuadrado latino no válido. Si un cuadrado se usa en un contexto booleano, debe evaluarse como `false` si ya ha agotado todas las posibilidades, y `true` si aún quedan cuadradados latinos por generar.
7. [5%] **Operador ++** (posincremento). Transforma un cuadrado latino en otro cuadrado latino distinto y válido pero retorna el cuadrado latino antes de hacer la modificación. Este método debe delegar responsabilidad en otros como `swapRows()` o `swapColumns()`. Además debe retornar una copia del objeto `LatinSquare`. ¿Soporta su clase la creación de copias de objetos?
8. [15%] **Sobrecargas fundamentales para manejo de memoria dinámica**. Dado que la clase `LatinSquare` utiliza memoria dinámica, debe evitar fugas de memoria o accesos inválidos a toda costa.
9. [10%] **Operador (i,j)**. Sobrecarga del operador paréntesis para acceder al elemento (i,j) del cuadrado latino. Debe tener sus dos variantes: acceso en modo sólo lectura y en modo escritura. Asume que los índices `i` y `j` son válidos. Es decir, si se utilizan valores fuera de rango hará que el programa se caiga.
10. [10% opcional] **Movimiento de memoria**. Haga su clase más eficiente la copia de objetos implementando los métodos que pueden aprovechar la memoria de otro objeto que está por eliminarse (`rvalue`).

Su clase `LatinSquare` debe hacer trabajar la siguiente función `main()`:

```
 1. #include "LatinSquare.h"
 2. #include <cstdlib>
 3.
 4. int main(int argc, char* argv[])
 5. {
 6.    if ( argc <= 1 ) { std::cout << "Usage: latinsquare SIZE\n"; return 0; }
 7.    size_t n = atoi(argv[1]);
 8.    if ( n == 0 ) { std::cout << "latinsquare: invalid size " << argv[1] << std::endl; return 1; }
 9.
10.    char symbols[n];
11.    for (size_t i = 0; i < n; ++i )
12.       symbols[i] = 'A' + i;
13.
14.    LatinSquare<char> square(n, symbols);
15.    for (size_t count = 0; square; ++square )
16.       std::cout << "Latin square " << ++count << ":\n" << square << std::endl;
17. }
```

Ejemplo de ejecución:

```
./latinsquare 3
Latin square 1:
A B C
B C A
C A B

Latin square 2:
B A C
C B A
A C B

Latin square 3:
C A B
A B C
B C A

Latin square 4:
C B A
A C B
B A C

Latin square 5:
A C B
C B A
B A C

Latin square 6:
C A B
B C A
A B C

Latin square 7:
B A C
A C B
C B A

Latin square 8:
B C A
A B C
C A B

Latin square 9:
C A B
A B C
B C A

Latin square 10:
A C B
B A C
C B A

Latin square 11:
B C A
C A B
A B C

Latin square 12:
B A C
C B A
A C B

Latin square 13:
B A C
A C B
C B A

Latin square 14:
A B C
C A B
B C A

Latin square 15:
C B A
B A C
A C B

Latin square 16:
C A B
B C A
A B C

$
```