

Prácticas de Programación 1

Grado en Ingeniería Informática



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

**Miguel Ángel Latre, Ricardo J. Rodríguez, Rafael Tolosana, José Luis Pina y
Javier Martínez**

Área de Lenguajes y Sistemas Informáticos
Departamento de Informática e Ingeniería de Sistemas



Universidad
Zaragoza

1542

Curso 2020-21

Práctica 5: Programas C++ con registros, matrices y matrices de registros


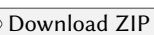
5.1. Objetivos de la práctica

En esta práctica se va a trabajar con registros (Tarea 5.2), matrices (Tarea 5.3) y matrices de registros (Tarea 5.4). Además, se va a hacer énfasis en la metodología de diseño descendente, puesto que la descomposición modular de los distintos problemas no se ofrece tan pautada como en prácticas anteriores.

Al igual que en las restantes prácticas, a la sesión asociada a esta práctica hay que acudir con el trabajo que se describe a continuación razonablemente avanzado, con objeto de aprovecharla con un doble objetivo:

- Consultar al profesor las dudas surgidas y las dificultades que hayan impedido resolver satisfactoriamente los problemas planteados y, con su ayuda, tratar de aclarar ideas y avanzar en la resolución de los problemas surgidos.
- Presentar al profesor el trabajo realizado para que este lo pueda revisar, advertir de posibles errores y defectos y dar indicaciones sobre cómo mejorarlo y, en su caso, corregirlo.

En el repositorio <https://github.com/progl-eina/practica5> tienes un área de trabajo para esta práctica, con la estructura de directorios y ficheros necesaria para que el fichero «Makefile» y las tareas de compilación, ejecución y depuración de Visual Studio Code funcionen adecuadamente.

Puedes descargarte el área de trabajo completa (botón   de la web del repositorio), y descomprimirla en tu directorio «Programacion1» como «practica5» (borra el sufijo «-master» que añade GitHub al preparar el fichero comprimido para su descarga).

5.2. Números complejos

Un número complejo es un número que puede ser expresado de la forma $a + bi$, donde a y b son números reales e i es la solución a la ecuación $x^2 = -1$. Como ningún número real satisface esta ecuación, a i se le llama unidad imaginaria. Dado el número complejo $a + bi$, se denomina parte real a a y parte imaginaria a b .¹

Escribe un programa que genere un vector de 5 números complejos, con partes real e imaginaria elegidas en un intervalo (x, y) de manera aleatoria. El programa debe, en primer lugar, solicitar los límites de este intervalo al usuario, garantizando que los valores introducidos por el usuario cumplen

¹Wikipedia contributors, «Complex number», *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Complex_number (accedido el 20 de noviembre de 2020).

que $x < y$. Después, mostrará al usuario todos los números complejos generados. Posteriormente, mostrará al usuario el número complejo de mayor módulo y el número complejo de menor módulo. Recuerda que el módulo de un número complejo $a+bi$ se calcula como $\sqrt{a^2 + b^2}$. El programa finalizará mostrando la suma de todos los números complejos generados. Recuerda que la suma de dos números complejos $a + bi$ y $c + di$ es $(a + c) + (b + d)i$. Todos los números decimales se mostrarán con una precisión de dos dígitos.

Para escribir este programa, **deberás definir un tipo de datos que permita representar números complejos y utilizar un diseño modular con, al menos, procedimientos o funciones para calcular lo siguiente:**

- La posición en el vector de 5 números complejos generados de manera aleatoria de aquel número complejo que tiene mayor módulo.
- La posición en el vector de 5 números complejos generados de manera aleatoria de aquel número complejo que tiene menor módulo.
- La suma de todos los números complejos contenidos en el vector.
- Escribir un número complejo en la pantalla.

Ejemplo de interacción:

```
Introduce los límites del intervalo (x < y): 3 -2
Introduce los límites del intervalo (x < y): -1 3.14

Números complejos generados:
-0.99+1.33i, -0.20+2.35i, 1.42+0.99i, 0.45+2.71i, 2.41+2.09i

Números complejos de mayor y menor módulo, respectivamente:
2.41+2.09i y -0.99+1.33i

Suma:
3.08+9.47i
```

Escribe el código de este programa utilizando el fichero «complejos.cpp» como código de partida, contenido en los ficheros de la práctica que te has descargado de GitHub.

Recuerda actualizar la cabecera del fichero y especificar todas las funciones que escribas.

Generación de números aleatorios

En informática, los números puramente aleatorios no existen, y por eso se les llama *números pseudoaleatorios*. Para generar números pseudoaleatorios en C++ puede utilizarse la función `rand()`² de la biblioteca `<cstdlib>`, que genera números **enteros** pseudoaleatorios entre 0 y `RAND_MAX`. Por ejemplo, el siguiente fragmento de código escribiría en pantalla 10 números pseudoaleatorios:

```
for (int i = 0; i < 10; i++) {
    cout << rand() << endl;
}
```

²Descripción de la función en <http://www.cplusplus.com/reference/stdlib/rand/>.

Si ejecutas el código anterior varias veces, comprobarás que la secuencia de números generada es siempre la misma. En el programa que se pide desarrollar, esto no es un problema. Sin embargo, si se desea que la secuencia de números pseudoaleatorios sea distinta cada vez, es preciso invocar a la función `srand()`³ previamente a la invocación a `rand()`. Normalmente, la función `srand()` se invoca una única vez al comienzo del programa.

Los números pseudoaleatorios se calculan a partir de unas funciones matemáticas que permiten ir obteniendo una secuencia de números. Cada vez que se llama a `rand()`, se obtiene el siguiente número de esa secuencia, usando para el cálculo el último número pseudoaleatorio que había calculado anteriormente. La función `srand()` nos permite establecer la *semilla* inicial, que es el primer número que se usará para generar esa secuencia de números. Esto garantiza que si conocemos la semilla inicial, la serie de números pseudoaleatorios será siempre la misma.

Observa el efecto de ejecutar el siguiente código varias veces:

```
srand(0xBAADBEEF);  
for (int i = 0; i < 10; i++) {  
    cout << rand() << endl;  
}
```

Como habrás comprobado, al inicializar la semilla al mismo número (en este caso al número entero `0xBAADBEEF`, expresado en notación hexadecimal), los números pseudoaleatorios que se generan son siempre los mismos.

La implementación de la función `rand()` es dependiente de la implementación. Por tanto, la calidad de la secuencia de números pseudoaleatorios que produce, así como su distribución y periodo, no están garantizadas. C++ dispone de una biblioteca predefinida denominada `<random>`, que es la que debería utilizarse en programas en los que la calidad de la pseudoaleatoriedad sea importante (no es el caso del programa que se solicita).

5.3. El Juego de la vida

El *Juego de la vida* es un autómata celular diseñado por el matemático británico John Horton Conway (1937–2020) que simula la evolución de una colonia de seres unicelulares sobre las celdas de un tablero rectangular. Cada celda puede estar o vacía o habitada por una sola célula viva. Cada celda, excepto las de la periferia, tiene ocho celdas vecinas: sus celdas adyacentes tanto vertical como horizontal y diagonalmente.

Cada cierto tiempo se produce una renovación generacional siguiendo las siguientes reglas:

- Una célula viva muere de inanición si en las celdas vecinas a la que ocupa hay menos de dos células vivas.
- Una célula viva sobrevive si en las celdas vecinas a la que ocupa hay 2 ó 3 células vivas.
- Una célula viva muere por superpoblación si en las celdas vecinas a la que ocupa hay más de tres células vivas.
- En una celda vacía puede nacer, por reproducción, una célula viva si en las celdas vecinas hay exactamente tres células vivas.

³Descripción de la función en <http://www.cplusplus.com/reference/cstdlib/srand/>.

Pese a que el juego original diseñado por John H. Conway tiene lugar en un tablero infinito, vamos a limitarnos a trabajar en un tablero de dimensiones máximas `MAX_FILAS`×`MAX_COLUMNAS`, siendo `MAX_FILAS` y `MAX_COLUMNAS` dos constantes definidas en el fichero «juego-vida.hpp» suministrado.

5.3.1. Trabajo a desarrollar en esta tarea

En esta tarea te pedimos que escribas dos programas relacionados:

- Un programa principal que implemente el *Juego de la vida* a partir de una configuración inicial parcialmente suministrada por el usuario.
- Un programa de pruebas que implemente el *Juego de la vida* a partir de unas configuraciones predeterminadas y que no va a requerir interacción por parte del usuario.

5.3.2. Programa principal

El programa principal que simula el *Juego de la vida* tiene que tener el siguiente comportamiento:

- El programa comienza solicitando al usuario el número de filas del tablero con el que desea trabajar. La solicitud se repite hasta que el usuario introduce un entero mayor estricto que 0 y menor o igual que `MAX_FILAS`.
- El programa solicita a continuación el número de columnas del tablero con el que desea trabajar. La solicitud se repite hasta que el usuario introduce un entero mayor estricto que 0 y menor o igual que `MAX_COLUMNAS`.
- El programa pide el número de generaciones a simular. La solicitud se repite hasta que el número introducido es mayor o igual que 0.
- A continuación, el programa inicializa las filas y columnas indicadas por el usuario de un tablero de forma aleatoria, de forma que en cada celda haya una célula con una probabilidad del 20 %.
- El programa debe escribir en la pantalla el estado inicial de ese tablero. Para ello, escribirá en la pantalla el mensaje “Generación 0” y, seguidamente, a partir de la línea siguiente, mostrará el tablero, a razón de una fila del mismo por línea. Las celdas vacías se representarán con un espacio en blanco y las celdas con una célula viva, con un asterisco (carácter ‘*’). Para que visualmente resulte más atractivo, cada una de las columnas de una celda se separará de la siguiente con un espacio en blanco.
- El programa continuará escribiendo en la pantalla los estados que va alcanzando el juego conforme pasa el tipo y se aplican las reglas de supervivencia descritas anteriormente. Para ello, escribirá el mensaje “Generación *i*”, donde *i* es el número de iteración y, a continuación, el estado del tablero según lo enunciado en el punto anterior.

En el fichero se proporciona el código de dos funciones que, utilizando *secuencias de escape*⁴, pueden servir para mejorar la apariencia gráfica de la simulación. Se sugiere utilizar antes de mostrar la generación inicial la función `borrarPantalla` una única vez, para borrar el texto escrito en el terminal y mover el cursor a la primera columna de la primera línea. La función `subirCursor` sube la posición del cursor el número de líneas determinado por su parámetro y se puede utilizar antes de escribir cada generación en la pantalla, para sobrescribir la escritura de la generación anterior.

⁴https://en.wikipedia.org/wiki/ANSI_escape_code

Si estas funciones no se ejecutaran correctamente en tu terminal porque este no soporta correctamente la secuencias de escape, simplemente no las utilices.

- El programa cesará su ejecución cuando se hayan mostrado en la pantalla tantos estados como generaciones haya indicado inicialmente el usuario o cuando se alcance un estado en el que no quede ninguna célula superviviente.

Si la simulación termina porque todas las células de la colonia han muerto, el programa debe indicarlo escribiendo en la pantalla el mensaje “Colonia extinguida.”. En caso contrario, debe escribir un mensaje indicando en número de células que quedan en la colonia.

A continuación se presenta un ejemplo de ejecución del programa:

```
Número de filas: -1
Número de filas: 80
Número de filas: 10
Número de columnas: -10
Número de columnas: 120
Número de columnas: 30
Número de generaciones: -6
Número de generaciones: 20
```

Generación 0

```
*  *          *      * *  * *  * * *
*      *          *      *      *
*  *  *          *      *      *
*      *  *  *      *      *  *  *
*      *      *      *      *  *
*      *      *  *  *      *  *  *
* *  *      *  *  *      *      *
*  *      *  *  *      *      *
*      *      *      *      *      *
*      *      *      *      *      *
*      *  *  *      *      *      *
```

...

Generación 20

```
*
* * * *
*      *  * * * *
*      *      *  * *  * *
*      *      *  * *  *  *
*      *      *      *
*      *      *      *      *
* *  *  *      *      *
*      *
```

Sobreviven 41 células.

5.3.3. Programa de pruebas

El programa de pruebas no requiere interacción con el usuario y su objetivo es simular el *Juego de la vida* con determinadas configuraciones iniciales predeterminadas:

- El programa comienza realizando una simulación de 30 generaciones de una configuración inicial consistente en un patrón denominado *bloque* y que es estable generación tras generación. Esta configuración inicial consiste en un tablero de dimensiones 4×4 con la configuración de celdas vacías y ocupadas que se muestra a continuación:

.	.	.	.
.	*	*	.
.	*	*	.
.	.	.	.

Nota: por razones de legibilidad, en los esquemas que se muestran aquí se visualizan las celdas vacías con el carácter '.'.

En esta fase, el programa de pruebas muestra una a una las 30 generaciones (que no varían en ningún momento) y finaliza indicando que sobreviven 4 células.

- En una segunda fase, el programa realiza una simulación de otras 30 generaciones de una configuración inicial correspondiente a un patrón oscilante denominado *intermitente*. Esta configuración inicial consiste en un tablero de dimensiones 5×5 con la siguiente configuración de celdas vacías y ocupadas:

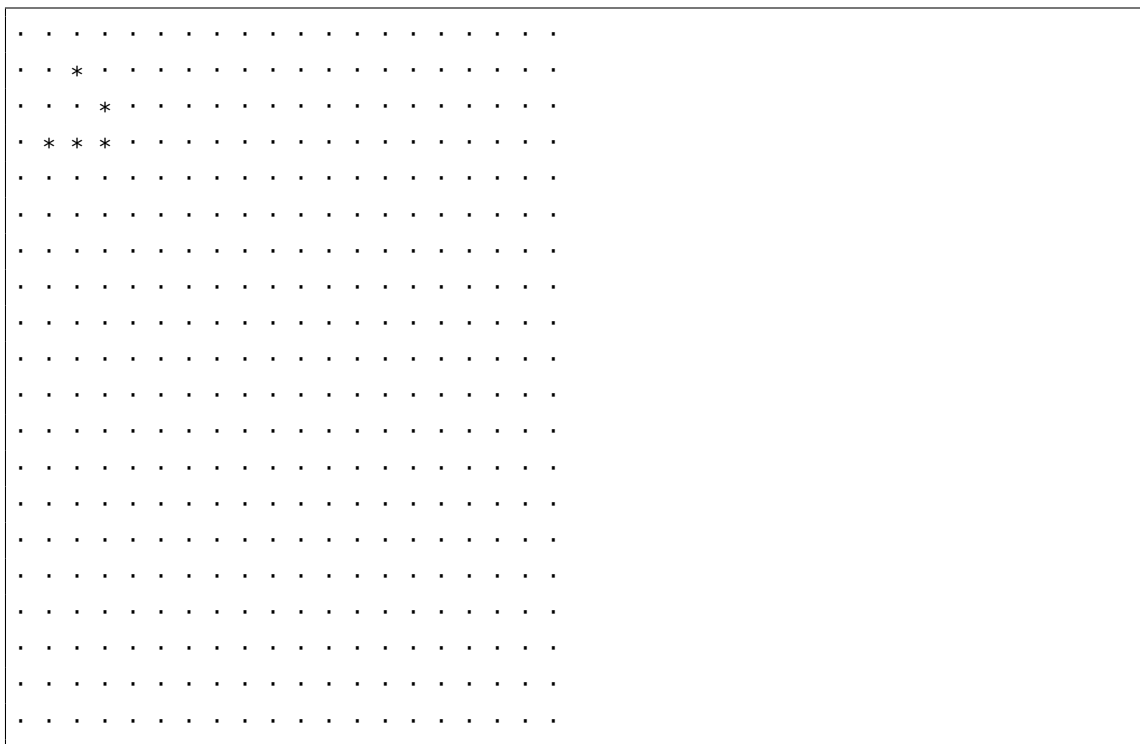
.
.
.	*	*	*	.
.
.

Esta fase debe finalizar con el programa indicando que sobreviven 3 células.

- Después, sigue una simulación de otras 30 generaciones de una configuración inicial correspondiente a otro patrón oscilante, denominado *faro*. Esta configuración inicial consiste en un tablero de dimensiones 6×6 con la siguiente configuración de celdas vacías y ocupadas:

.
.	*	*	.	.	.
.	*	*	.	.	.
.	.	.	*	*	.
.	.	.	*	*	.
.

- Por último, se realiza una simulación de 60 generaciones de una configuración inicial correspondiente a otro patrón móvil, denominado *deslizante*. Esta configuración inicial consiste en un tablero de dimensiones 20×20 con la siguiente configuración de celdas vacías y ocupadas:



La simulación debe terminar indicando que sobreviven 5 células.

Todas estas simulaciones se ejecutan una detrás de otra, sin intervención directa del usuario, tal y como se ha comentado al inicio de la sección.

En la página de la Wikipedia en inglés tienes información sobre otros patrones que puedes incluir voluntariamente en este programa de pruebas (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life#Examples_of_patterns).

5.3.4. Algunas notas sobre esta tarea

En esta tarea, se solicita la escritura de dos programas distintos que comparten mucha funcionalidad. **Utiliza una metodología de diseño descendente para estructurar el código en distintas funciones de no muchas líneas cada una. Distribuye el código en tres módulos distintos:**

1. El módulo principal del programa interactivo que simula el *juego de la vida*. Este módulo incluirá la función `main()` de dicho programa, así como todas las funciones que necesite solo este programa, como son las que interactúan con el usuario pidiéndole los datos iniciales de configuración o el código que inicializa aleatoriamente el tablero del juego.

Ubica este módulo en el fichero «juego-vida-main.cpp»

2. El módulo principal del programa de pruebas que realiza simulaciones a partir de patrones concretos del juego de la vida. Este módulo incluirá la función `main()` de este segundo programa, así como todas las funciones que necesite solo este programa, como puedan ser funciones que realicen las inicializaciones específicas de los tableros correspondientes a los distintos patrones.

Ubica este módulo en el fichero «juego-vida-test.cpp»

3. El módulo «juego-vida». Coloca en este módulo el código común a ambos programas. En el fichero «juego-vida.cpp» ubica todo el código de las funciones comunes y en el fichero «juego-vida.

hpp», escribe las cabeceras de aquellas que luego se vayan a utilizar en los módulos principales, así como las declaraciones de las constantes que sean comunes a ambos.

Recuerda actualizar las cabeceras de los ficheros y especificar todas las funciones que escribas.

Pista 1: Al calcular la generación siguiente a una dada, todos los nacimientos y muerte de células deben ocurrir simultáneamente. Por ello, deben utilizarse dos tableros: uno contiene la generación actual, y el otro la siguiente generación.

Pista 2: Al calcular el número de células vivas vecinas a una celda dada, se puede evitar la complejidad de revisar los casos particulares correspondientes a las celdas periféricas (que tienen menos de 8 vecinas) utilizando una función que tenga el siguiente comportamiento: dado el tablero, las dimensiones del mismo y las coordenadas (i, j) de una celda (válidas o no), esta función devolvería el valor booleano `false` en el caso de que las coordenadas no fueran válidas (por ser una de ellas o ambas menores que 0 o mayores que la dimensión máxima correspondiente). En caso contrario (es decir, cuando (i, j) representaran una celda válida del tablero), esta función devolvería `true` si la celda válida contiene una célula o `false` si está vacía.

5.4. Modificación de imágenes de mapas de bits (BMP)

En informática, una forma de representar imágenes es a través de *mapas de bits*: la imagen se representa a través de una matriz de puntos discretos, denominados *píxeles* (del acrónimo en inglés *picture element*). Cada píxel representa la menor unidad en la que se puede descomponer la imagen y tiene un color determinado y homogéneo. Este color se representa habitualmente a través de una combinación de diferentes intensidades de luz roja, verde y azul. Así, por ejemplo, un píxel de color blanco se representa a través de la combinación de intensidades de luz roja, verde y azul máxima; un píxel negro, a través de la combinación de intensidades de luz roja, verde y azul nula; un píxel amarillo a través de la combinación de intensidades de luz roja y verde máximas y de luz azul nula; y un píxel gris a través de una combinación de intensidades de luz roja, verde y azul intermedia. Habitualmente, el nivel de intensidad de cada canal de luz se representa a través de un entero positivo entre 0 y 255, donde 0 representa una intensidad nula y 255 una intensidad máxima.

El formato *Windows bitmap* (BMP) es un formato de fichero para almacenar imágenes de mapas de bits, utilizado sobre todo en el sistema operativo Windows. En estos ficheros se almacenan, entre otros datos, la matriz de píxeles que forman la imagen en sí (que se almacena por filas), así como el número de píxeles de cada fila (altura de la imagen) y de cada columna (anchura de la misma).

En esta tarea, vamos a trabajar con ficheros BMP de 24 bits con una altura y anchura múltiplos de 4 y menores de 800 píxeles.

En el módulo «imagen-bmp» os facilitamos las definiciones de dos tipos registro denominados `Pixel` e `Imagen`. El tipo `Pixel` permite representar píxeles a través de tres campos denominados rojo, verde y azul. Como sólo es necesario representar enteros entre 0 y 255, utilizamos el tipo `char`, que se codifica con 8 bits (aunque no nos interesan sus valores como carácter, sino sus valores enteros).

El tipo `Imagen` permite representar el contenido completo de un fichero de imagen BMP: su anchura, altura y la matriz de píxeles. Se incluyen también dos campos adicionales para representar la información de la cabecera.

En el mismo módulo proporcionamos dos funciones que permiten leer un fichero de tipo BMP (función `leerImagen()`) y almacenar su contenido en un registro `Imagen` y almacenar el contenido de un registro `Imagen` en un fichero con formato BMP (función `guardarImagen()`). El código de estas funcio-

nes lo utilizaremos de ejemplo en clase de teoría en el tema dedicado a ficheros binarios.

En el fichero «imagen-bmp-main.cpp» es donde escribirás el código correspondiente al problema que se plantea a continuación. El fichero que te has descargado de GitHub contiene un ejemplo en el que se utilizan las funciones de lectura y escritura de imágenes mencionadas antes. Ejecútalo para comprobar que todo funciona correctamente. Si es así, verás como se genera en la carpeta «datos» una copia del fichero «prog1.bmp» denominada «imagen-generada.bmp».

Observa que en la función main de este fichero se ha declarado una variable de tipo Imagen como *estática*, pese a que se trata de un concepto con el que no se ha trabajado en el curso. La razón de su utilización es el hecho de que la variable `img` es demasiado grande para ser almacenada en la pila (el segmento de memoria donde se ubican habitualmente las variables locales de una función). Con el modificador `static` se indica al compilador que la ubique en otro segmento de memoria (concretamente, en el segmento de datos). Conocerás más detalles sobre los distintos segmentos de memoria en la asignatura *Arquitectura de Computadores* en el 2.º cuatrimestre.

Se pide un programa que tenga el siguiente comportamiento:

- El programa pide el nombre de un fichero con formato BMP. Como los ejemplos que os hemos pasado están en una carpeta denominada «datos», si el usuario quiere utilizar uno de ellos, su respuesta debe comenzar con la cadena «datos/». **No es responsabilidad del programa comprobar este aspecto.**
- Si el fichero puede leerse correctamente, el programa solicita una opción a través de un carácter: 'N' para negativizar (invertir los colores de una imagen) y 'R' para girar la imagen 180°. Debe ser indiferente que el usuario escriba la opción en mayúsculas o minúsculas.
- El programa pide el nombre del fichero en el que guardar la imagen transformada y finalmente la guarda.

Para negativizar una imagen, hay que invertir la intensidad de cada canal de luz de cada píxel de la imagen. Para ello, si x es la intensidad de uno de los canales (rojo, verde o azul) expresada como un entero entre 0 y 255, la expresión $255 - x$ representa la intensidad opuesta.

Si la imagen original fuese la del fichero «datos/unizar.png», la imagen negativizada sería la siguiente:



Para girar una imagen 180°, sólo es necesario cambiar los datos de la matriz $\text{ancho} \times \text{alto}$ donde se guardan los píxeles: el elemento (i, j) de la matriz pasará a ser el $(\text{alto} - 1 - i, \text{ancho} - 1 - j)$.

Si la imagen original fuese la del fichero «datos/prog1.png», la imagen rotada sería la siguiente:



Prog1

Se muestran a continuación dos ejemplos de interacción del programa:

Escriba el nombre de un fichero BMP: datos/unizar.bmp
Imagen "datos/unizar.bmp" leída con éxito.
Escriba una opción (N - negativizar; R - rotar): N
Escriba el nombre del fichero destino: datos/negativo.bmp
Imagen "datos/negativo.bmp" creada con éxito.

Escriba el nombre de un fichero BMP: datos/prog1.bmp
Imagen "datos/prog1.bmp" leída con éxito.
Escriba una opción (N - negativizar; R - rotar): r
Escriba el nombre del fichero destino: datos/rotada.bmp
Imagen "datos/rotada.bmp" creada con éxito.

Si quieres realizar pruebas de este programa con imágenes distintas a las que os hemos facilitado, puedes utilizar el programa Paint de Windows. Asegúrate de guardar la imagen como una imagen BMP (Archivo » Guardar como... » Imagen BMP » Mapa de bits de 24 bits (*.bmp;*.dib)) y de que la altura y la anchura sea menores de 800 y ambas múltiplo de 4.

5.5. Entrega de la práctica

Debes de realizar una planificación del trabajo que se pide realizar en esta práctica como para poder entregarla el sábado 5 de diciembre. No obstante, se fija como fecha de entrega el sábado 12 de diciembre. **Esta fecha no será prorrogada**, máxime cuando tenemos previsto publicar el enunciado de la práctica 6 el día 4 de diciembre.

Antes del **sábado 12 de diciembre a las 18:00**, se deberán haber subido a Moodle los siguientes ficheros:

- «complejos.cpp»
- «juego-vida.cpp»
- «juego-vida.hpp»
- «juego-vida-main.cpp»
- «juego-vida-test.cpp»
- «imagen-bmp-main.cpp»

Los ficheros «imagen-bmp.cpp» e «imagen-bmp.hpp» no hay que subirlos a Moodle, puesto que no deben ser modificados (ni añadiendo funciones en el mismo, ni modificando las cabeceras de las ya existentes). Si los modificas, es muy probable que el resultado de la corrección que realicemos los profesores del programa «imagen-bmp-main.cpp» termine en error de compilación, lo que acarreará una calificación de 0 en esa parte.