Programe la función *comprimir* con el siguiente encabezado:

```
typedef unsigned int uint;
uint comprimir(uint *a, int nbits);
```

Esta función comprime múltiples enteros sin signo almacenados en el arreglo *a* en un solo entero sin signo. Para ello se retorna la concatenación de todos los elementos en *a* truncados a *nbits*. La cantidad de elementos almacenados en el arreglo *a* es el número de enteros de *nbits* que caben en un entero sin signo, es decir el máximo *k* que cumple con *k*nbits<=sizeof(uint)*8*, en donde *sizeof(uint)*8* es el tamaño de un entero sin signo (no es 32 en algunas plataformas).

Ejemplo de uso:

```
uint a[]= { 0b 100 110 101 011 000, 0b 000 101 101 011,
            0b 100 001 010 000 };
uint r= comprimir(a, 9);
// r es 0b 101 011 000  101 101 011  001 010 000
//         <- a[0]  ->  <- a[1]  ->  <- a[2]  ->
```

Observe que al truncar *a[0]* a 9 bits se perdieron los bits más significativos de *a[0]*. Con el fin de facilitar la comprensión se empleó la notación *0b...* para expresar números en base 2, pero no es parte del lenguaje C. Considere que el parámetro *nbits* puede variar entre 1 y *sizeof(uint)*8*.

Restricciones:

- Ud. no puede usar los operadores de multiplicación, división o módulo (* / %). Use los operadores de bits eficientemente.
- Si necesita calcular el número de bits en variables de tipo *uint*, calcule *sizeof(uint)<<3*. La cantidad de bits en un byte es siempre 8.
- Se descontará medio punto por no usar el estilo de indentación de Kernighan como se explica en esta sección de los apuntes.
- El estándar de C no especifica el resultado para desplazamientos mayores o iguales al tamaño del operando. Sanitize rechaza el desplazamiento *x<<nbits* cuando *nbits* es 32 o superior. En esta tarea use *x<<(nbits-1)<<1* porque sí va a funcionar considerando las restricciones en el rango que puede tomar *nbits* en esta tarea.

Instrucciones

Baje *t1.zip* de U-cursos y descomprímalo. El directorio *T1* contiene los archivos (a) *test-comprimir.c* que prueba si su tarea funciona y compara su eficiencia con la solución del profesor, (b) *prof.ref-x86_64 y prof.ref-aarch64* con los binarios ejecutables de la solución del profesor, (c) *comprimir.h* que incluye el encabezado de la función pedida, y (d) *Makefile* que le servirá para compilar y ejecutar su tarea. Ud. debe programar la función *comprimir* en el archivo *comprimir.c*. Se incluye una plantilla en *comprimir.c.plantilla*.

Pruebe su tarea bajo Debian 11 de 64 bits nativo o virtualizado con VirtualBox, Vmware, QEmu o WSL 2. **Ejecute el comando *make* sin parámetros**. Le mostrará las opciones que tiene para compilar su tarea. Estos son los requerimientos para aprobar su tarea:

- *make run* debe felicitarlo por aprobar este modo de ejecución. Su solución no debe ser 80% más lenta que la solución del profesor.
- *make run-g* debe felicitarlo.
- *make run-san* debe felicitarlo y no reportar ningún problema como por ejemplo desplazamientos indefinidos.

Cuando pruebe su tarea con *make run* asegúrese que su computador esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo. De otro modo podría no lograr la eficiencia solicitada.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *comprimir.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.