Parte a.- Programe la siguiente función: `void reducir(char *s);`

Esta función reemplaza múltiples espacios contiguos en el string *s* por un solo espacio. Ejemplo de uso:

```
char s[]= "   hola     que     tal   ";
reducir(s); //s es " hola que tal "
```

Observe que reducir modifica el mismo string que recibe como parámetro. Su solución debe ser eficiente. No puede pedir memoria adicional con *malloc* o declarando un arreglo.

Parte b.- Programe la siguiente función: `char *reduccion(char *s);`

Esta función retorna un nuevo string construido a partir de *s* reemplazando los múltiples espacios contiguos por un solo espacio. Ejemplo de uso:

```
char *r= reduccion("   hola     que     tal   ");
                          //r es " hola que tal "
```

Observe que reduccion *no modifica* el string que recibe como parámetro. Su solución debe ser eficiente. Se le permite invocar una sola vez *malloc* para pedir el espacio necesario para el nuevo string y *debe pedir exactamente el espacio requerido para el resultado*. No puede pedir más memoria con *malloc* o con un arreglo. Por lo tanto, antes de invocar *malloc* necesitará contabilizar la cantidad de caracteres del resultado.

Restricciones de estilo para ambas partes: Ud. no puede usar el operador de subindicación [ ] ni su equivalente **(p+i)*. Para recorrer el string use los operadores ++ -- += ... etc. Use múltiples punteros para direccionar distintas partes de los strings. Esta restricción busca que Ud. aprenda el estilo de uso de punteros en C.

Restricciones de eficiencia: La función *reducir* debe tomar menos tiempo que 0.9 veces el tiempo que toma la función *reduccion*. El test de prueba de la tarea incluye una versión de *reduccion* implementada a partir de su versión de *reducir*. Su versión de *reduccion* debe tomar menos tiempo que 0.9 veces el tiempo que toma la versión incluida en el test de prueba. Un objetivo secundario de esta tarea es mostrar que el uso de *malloc* tiene un costo en tiempo de ejecución.

Instrucciones

Baje *t2.zip* de U-cursos y descomprímalo. El directorio *T2* contiene los archivos (a) *test-t2.c*, que prueba si su tarea funciona, (b) *t2.h* que incluye los encabezados de las funciones pedidas, y (c) *Makefile* que le servirá para compilar su tarea. Ud. debe crear un archivo *t2.c* y programar ahí las funciones pedidas. Compile su tarea con el comando *make* sin parámetros. Depure su tarea con el comando *ddd test-t1*.

A partir de esta tarea es obligatorio que Ud. pruebe su tarea en la distribución Debian de Linux con soporte para punteros de 32 y 64 bits. Debian de 64 bits bajo WSL 2 sirve para este propósito, así como la máquina virtual oficial del curso con Debian para VirtualBox publicada en: *https://users.dcc.uchile.cl/~lmateu/CC3301/*

Una vez que su tarea funcione correctamente en la plataforma de su elección, cree un archivo *resultados.txt* y pruebe su tarea bajo Debian con los comandos *make test-g*, *make test-O* y *make test-O-m32*, anotando en *resultados.txt* los tiempos que toman las funciones *reducir*, *reduccion* y *reduccion* a partir de *reducir*. Estos tiempo son reportados por el programa de prueba (*test-t1*). La compilación no puede arrojar errores o warnings y la ejecución debe terminar mostrando "Felicitaciones ..." para las 3 ejecuciones con *make*.

Entrega

Ud. solo debe entregar los archivos *t2.c* y *resultados.txt* en el formato *.zip* por medio de U-cursos. Se descontará medio punto por día de atraso. No se consideran los días de vacaciones, sábado, domingo o festivos.