Parte a.- Programe la siguiente función:

```
int aislar_palabras(char *str);
```

Esta función transforma el string *str* dejando solo las palabras contenidas en *str* que estén formadas por caracteres alfabéticos. Además retorna el número de palabras encontradas. Las palabras quedarán en el mismo string *str* separadas por un espacio en blanco. Este es un ejemplo de uso:

```
char str[]=
 "  return   ( 'a'<=ch && ch<='z') || ('A'<=ch && ch<='Z');";
int n= palabras(str);
// n=9 y str es "return a ch ch z A ch ch Z";
```

Ayuda: Note que el string del ejemplo le servirá para determinar si un carácter es alfabético. Declare 2 punteros. Use uno para recorrer los caracteres del string y el otro para ir almacenando los caracteres que permanecen en el string.

Restricciones: Ud. no puede usar el operador de subindicación [ ], ni su equivalente **(p+i)*. Use ++ -- *p+i* o *p-i*. Por razones de eficiencia, Ud. no puede usar *malloc* o declarar un arreglo para pedir memoria adicional.

Parte b.- Programe la siguiente función:

```
char *palabras(char *str);
```

Esta función retorna un nuevo string construido a partir de *str* dejando solo las palabras contenidas en *str* que estén formadas por caracteres alfabéticos. Ejemplo de uso:

```
char *r= palabras("  speed * time + dist  ");
                          //r es "speed time dist"
```

Observe que *palabras no modifica* el string que recibe como parámetro.

Restricciones*:* Ud. no puede usar el operador de subindicación [ ], ni su equivalente **(p+i)*. Use ++ o *p+i*. Para recorrer el string use aritmética de punteros. Use *malloc* para pedir memoria para el string resultante. Debe pedir exactamente la cantidad de bytes que necesita el resultado, no más. Para el ejemplo debe pedir 16 bytes. Si pide más memoria que la que necesita, el test de uso de memoria podría agotar la memoria de su computador haciendo que se ponga muy lento antes de que el programa falle.

Ayuda: Use un primer *malloc* para crear una copia de *str*. Luego use *aislar_palabras* de la parte *a*. Llame por segunda vez *malloc* para pedir memoria para el resultado y copie ahí el resultado de *aislar_palabras*. Libere la memoria pedida en el primer llamado a *malloc*.

Instrucciones

Baje *t2.zip* de U-cursos y descomprímalo. El directorio *T2* contiene los archivos (a) *test-aislar.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) *aislar.h* que incluye los encabezados de las funcion pedidas, y (d) *Makefile* que le servirá para compilar y ejecutar su tarea. **Ejecute en un terminal el comando *make*** para recibir instrucciones adicionales. 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 goteras de memoria.

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 *aislar.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.