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. Por razones de eficiencia, en esta función Ud. no puede pedir más memoria con *malloc* o declarar un nuevo arreglo de caracteres. 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.

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. 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: Ninguna de sus funciones puede tomar más del doble del tiempo que toma la solución eficiente del profesor.

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-t2*.

Ud. debe probar su tarea bajo *Debian 10* de 64 bits con los comandos (i) *make* que compila con opciones de depuración, (ii) *make test-O* que compila con opciones de optimización y compara la eficiencia de su solución con la del profesor y (iii) *make test-vg* que usa *valgrind* para verificar el uso correcto de *malloc*. Bajo Debian, *valgrind* se instala con el comando: *sudo apt-get install valgrind*

Su tarea será aprobada cuando el comando *make* termine mostrando el mensaje:

```
Felicitaciones: su solucion es correcta
```

Además el comando *make test-O* debe terminar mostrando el mensaje:

```
Felicitaciones: su solucion es correcta y eficiente
```

Y la ejecución con make test-vg debe mostrar:

```
Felicitaciones: su solucion es correcta
ERROR SUMMARY: 0 errors from 0 contexts
```

Entrega

Ud. solo debe entregar el archivo *t2.c* por medio de U-cursos. Se descontará medio punto por día de atraso. No se consideran los días de vacaciones, sábado, domingo o festivos.