-Algoritmos de las funciones y programas de prueba:

Desarrollo1:

1.1: pseudo-código read_integer:

```
        leer primer caracter
                si caracter = ENTER
                        entonces termina
                fsi
                si caracter = signo negativo
                        entonces boolean = true
                fsi
                else caracter != signo negativo
                        entonces branch numero

Bucle:
        leer por teclado
                si caracter = ENTER
                        entonces termina
                fsi
numero:         si caracter != digito
                        vuelve a bucle
                fsi
        convertir caracter a numerico
        multiplicar la variable contadora por 10
        acumular caracter nuevo en variable
        volver a bucle

fin_Bucle:

        guarda en registro y finaliza programa
```

Desarrollo 2:

2.1: pseudo-código print_integer:

```
        puntero a variable CADENA
        puntero al final y añadimos carácter nulo

                si el numero es negativo (mascara)
                        entonces booleano = true
                                        convertimos numero a positivo
                fsi

Bucle:
        si numero < 10
                fin_Bucle
                dividir numero entre 10
                obtener resto y convertirlo en carácter
                almacenarlo de forma correspondiente en la variable CADENA
                actualizar el puntero de forma decreciente
Fin_Bucle:

        si numero es negativo (booleano)
                entonces añade signo "-" en cadena
                        convertimos numero en negativo
```

fsi

fin: proyectar cadena de caracteres por pantalla.

3. Programas de Prueba:

3.1: prueba_integer:

Invocar función read_integer
fin

3.2: prueba_write:

Num = -123
Invocar función print_integer
fin

-Anotaciones de cara al desarrollo de la práctica:
Las dos funciones previamente diseñadas responden a sus requerimientos principales: leer una serie de caracteres numéricos codificados en ASCII para convertirlos en un numero entero y viceversa. Su funcionalidad se complementa con las pruebas realizadas en cada una de ellas estudiando los casos especiales como el cero, números negativos, números no numéricos, extremos, primer caracter siendo numero en read_integer, etc.

-Cuestiones a responder:
1: para la programación modular ha sido necesario añadir la directiva .extern en el código fuente. De esta forma, la función diseñada podrá ser utilizada en futuros programas.

2: con objetivo de generar un ejecutable en base a programación modular es imprescindible hacer una llamada al fichero objeto de la función que queramos usar en el momento de lincado.
En esta practica hemos utilizado: ld -Ttext=18088 -o prueba_write prueba_write.o **print_integer.o**

3: en el primer desarrollo de la práctica surgieron errores de sintaxis (fallo ensamblaje) como por ejemplo el “#” a la hora de declarar constantes porque no se puede utilizar. Por consiguiente se ha procedido a regresar al código, corregirlo y volverlo a ensamblar.
En el segundo desarrollo surgieron una serie de errores funcionales que se fueron resolviendo a medida que se depuraba el código con detalle. En este caso, la dificultad recae más en la localización del fallo que su propia resolución.