Pregunta 1

Programe la función: `void alinear_der(char *str);`

Esta función modifica el string *str* de modo que quede alineado a la derecha. Es decir mueve todos los espacios en blanco al final de *str* al comienzo (su tamaño se preserva). Ejemplo:

```
char s1[]= "hola que tal   ";
alinear_der(s1); // s1 es "   hola que tal"
char s2[]= "  ";
alinear_der(s2); // s2 es "  " (no se modifica)
```

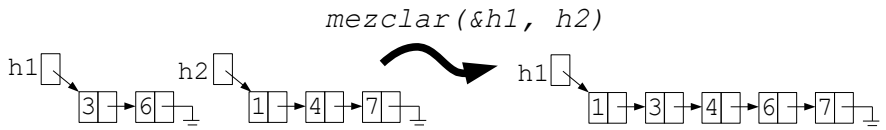
Restricciones: Ud. no puede usar el operador de subindicación [ ], ni su equivalente **(p+i)*. Para recorrer el string use aritmética de punteros como *p*++, *p*-- o *p*+*i*.

Ayuda: Use múltiples punteros para direccionar distintas partes del string. Posiciónese al final del string y regrese buscando el primer caracter que no sea un espacio.

Pregunta 2

```
typedef struct nodo {
  int x;
  struct nodo *prox;
} Nodo;
void mezclar(Nodo **ph1,
             Nodo *h2);
```

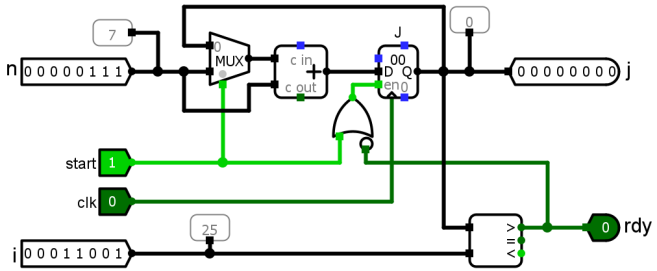
Programe la función *mezclar* que une 2 listas simplemente enlazadas ordenadas ascendentemente. La lista resultante también debe quedar ordenada ascendentemente. El encabezado es el que se indica en el cuadro de la derecha. La siguiente figura muestra el resultado de invocar *mezclar(&h1, h2)*. Los punteros *h1* y *h2* son de tipo *Nodo**.



Restricciones: Ud. no puede usar ninguna forma de iteración (while, for, etc.). Ud. debe usar recursividad (¡la versión no recursiva es complicada!). Ud. no puede usar *malloc*. Reutilice los nodos de los argumentos.

Pregunta 3

Parte a.- (2 puntos) La figura muestra un circuito con entradas *start*, *clk*, *i*, *n* y salidas *rdy* y *j*. La entrada *n* es 7 y *i* es 25. Ambas se mantienen constantes. Un ciclo del reloj inicia con el cambio de 1 a 0 de *clk* y termina con el siguiente cambio de 1 a 0 de *clk*. En el ciclo 1 del reloj *start* se pone en 1, en el ciclo 2 *start* se pone en 0 y luego se mantiene constante. Indique los valores de las salidas *rdy* y *j* en los ciclos 1, 2, 3, 4, 5, 6 y 7.



Parte b.- (4 puntos) Traduzca la función de la derecha a assembler Risc-V. Optimice el código en assembler para reducir la cantidad de instrucciones.

```
int menores(int *a, int n, int x) {
  int *p= a;
  int *q= a;
  for (int i= 0; i<n; i++){
    int y= *(p++);
    if (y<x)
      *(q++) = y;
  }
  return q-a;
}
```

Pregunta 4

I. (1,5 puntos) La función *mult* de la derecha calcula en paralelo el producto de los elementos de los vectores *a* y *b*, cada uno de *n* elementos, dejando el resultado en el vector *c*. Esta función contiene 3 errores de programación relacionados con los procesos de Unix. Indique cuáles son.

```
void mult(double *a, int n,
     double *b, double *c) {
  int h= (n+1)/2;
  if (fork()==0) {
    for (int i= 0; i<h; i++)
      c[i] = a[i]*b[i];
  }
  else {
    for (int i= h; i<n; i++)
      c[i] = a[i]*b[i];
  }
}
```

II. (4,5 puntos) Corrija los errores de la función *mult*. Su función debe invocar una sola vez *fork* para paralelizar la multiplicación para un computador con 2 cores.