Pregunta 1

Programe la función: *void* ***corregir****(char *s)*. Esta función elimina todos los espacios que precedan a una coma (caracter ‘,’). Ejemplo de uso:

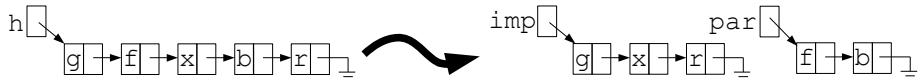
```
char s[]= "  h o, l  a  ,  q   , u  e t     ,a  l";
corregir(s); // s es "  h o, l  a,  q, u  e t,a  l"
```

Restricciones: No puede usar las funciones de manejo de strings como *strcmp*, *strncmp*, *strcpy*, *strlen*, etc. No use el operador de subindicación de arreglos [ ] ni su equivalente **(p+i)*, use aritmética de punteros. No puede pedir memoria adicional con *malloc* ni declarar arreglos. Necesitará usar punteros adicionales.

Pregunta 2

Programe eficientemente la función *dividir* que descompone una lista en 2 sublistas. Debe tener el encabezado de la derecha. Esta función deja en **pimpares* una lista compuesta por el primer, tercer, quinto, etc. nodo en *lista*. En **ppares* quedan los nodos segundo, cuarto, sexto, etc. de *lista*. Ambas listas deben terminar con el nodo *NULL*. En el siguiente ejemplo de uso las variables *h*, *imp* y *par* son de tipo *Nodo**:

```
typedef struct nodo {
  char c;
  struct nodo *prox;
} Nodo;
void dividir(
    Nodo *lista,
    Nodo **pimpares,
    Nodo **ppares );
```



dividir(h, &imp, &par);

Restricción: No puede usar *malloc*. Debe reutilizar los nodos que recibe en *lista*.

Ayuda: si usa recursión puede llegar a una solución con muy pocas líneas.

Pregunta 3

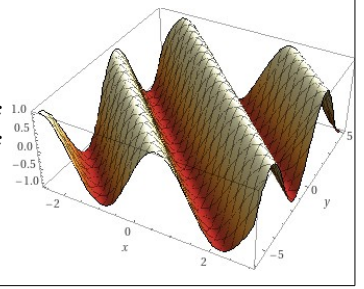
El programa en assembler Risc-V al inicio de la columna derecha es el resultado de compilar la función *incognito*. Programe la función *equivalente* a *incognito* en C sin usar la instrucción **goto** de C. Preocúpese de *reproducir* en C todos los aspectos de la función original en assembler, en particular el valor retornado.

```
incognito:
        mv      a5,a0
        mv      a0,a1
        sw      a2,0(a5)
        slli    a4,a1,2
        add     a5,a5,a4
.L2:
        addi    a0,a0,-1
        addi    a5,a5,-4
        lw      a4,0(a5)
        bne     a4,a2,.L2
        ret
```

Pregunta 4

La función *volumen* de más abajo estima el volumen bajo la superficie descrita por la función *f(x,y)* a partir de *n* evaluaciones de *f* en puntos pseudo-aleatorios. Más precisamente el volumen determinado por el rectángulo con vértices (x_i, y_i, 0) y (x_f, y_f, 0) y la superficie descrita por los puntos (x, y, *f(x,y)*) con x e y variando en los intervalos [x_i, x_f] e [y_i, y_f].

```
double volumen( double (*f)(double x, double y),
    double xi, double xf, double yi, double yf,
    int n, int p) {
  double suma= 0;
  for (int k= 0; k<n; k++) {
    double x= double_random(xi, xf);
    double y= double_random(yi, yf);
    suma += (*f)(x, y);
  }
  return (xf-xi)*(yf-yi)*(suma/n);
}
```



La figura muestra la superficie descrita por *sin(2x+y)* al variar x en [-2.5, 3] e y en [-6, 5]. El volumen pedido es el que queda bajo esta superficie hasta la coordenada z=0.

Reprograme esta función para que el cálculo se realice en paralelo usando p cores. Para ello Ud. *debe* usar *fork* para crear p nuevos procesos pesados. Cada proceso evalúa la función f en n/p puntos calculados pseudo-aleatoriamente. Use un *pipe* para que cada proceso hijo entregue su suma al padre. El padre solo espera a los hijos y luego calcula la estimación del volumen a partir de las sumas calculadas por los hijos. Por simplicidad considere que n es múltiplo de p. Para evitar que los hijos evalúen f en los mismos puntos, Ud. *debe* invocar *srandom(getUSecsOfDay()*getpid())* al comenzar cada hijo.