Pregunta 1

Programe la función: *Decimal* ***sumaDecimal****(Decimal x, Decimal y);*

El tipo *Decimal* es un entero sin signo de 64 bits que almacena números en *formato decimal* de hasta 16 dígitos. Esto significa que cada dígito de *x* se representa con 4 bits de *x*. Por ejemplo el número 15 se representa en formato decimal como 0x15 (en binario: … 0001 0101), a pesar de que 0x15 representa en formato binario el número 21. La función *sumaDecimal* recibe 2 números *x* e *y* en formato decimal y retorna su suma en formato decimal.

Ejemplos de uso: *sumarDecimal(0x**37**,0x**45**)* es 0x**82** porque **37**+**45** es **82** y *sumarDecimal(0x68, 0x45)* es 0x113. Observe que usar el operador + para hacer la suma no sirve porque 0x37 + 0x45 es 0xad, no 0x82. Este es un corto video que le recordará como sumar números en base 10. Ignore el caso en que el resultado no es representable con 16 dígitos.

Restricciones: No use los operadores de multiplicación, división o módulo (* / %). Use eficientemente los operadores de bits, sumas y restas.

Pregunta 2

Programe la siguiente función: *char* ***sumarStr****(char *a, char *b);*

Esta función recibe 2 strings ***del mismo largo*** que almacenan números en base 10, ***sin signo ni espacios en blanco***. Debe sumar *b* al parámetro *a*. Por ejemplo si *a* es igual a "068", después de invocar *sumarStr(a, "045")*, *a* debe ser "113". Normalmente el valor retornado debe ser el caracter '0', pero si el resultado de la suma no es representable en el largo de *a*, debe retornar '1'. Por ejemplo si *a* es igual a "68", después de invocar *sumarStr(a, "45")*, *a* debe ser "13" y valor retornado será el caracter '1'.

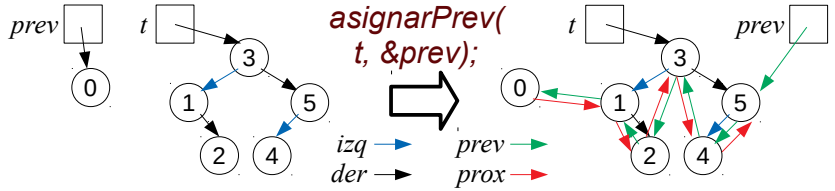
Restricciones: No use el operador de subindicación de arreglos [ ] ni su equivalente **(p+i)*, use aritmética de punteros. No puede pedir memoria adicional usando *malloc* o declarando un arreglo de caracteres. Necesitará declarar punteros adicionales.

Ayuda: Dado el caracter *c* en ASCII (por ejemplo '3'), la expresión *c*-'0' entrega su valor numérico (por ejemplo '3'-'0' es el entero 3). Dado un entero *d* entre 0 y 9 (por ejemplo 7), el caracter que representa en ASCII ese número es d+'0' (por ejemplo 7+'0' es '7'). **No funciona *atoi(a)*** porque *a* puede ser muy grande.

Pregunta 3

En un *recorrido en orden* de un árbol binario, se visita recursivamente primero el subárbol izquierdo, luego se visita la raíz y finalmente se visita recursivamente el subárbol derecho. Considere que se está visitando un nodo T al recorrer un árbol en orden. Se define como *previo* a T el nodo que se visitó anteriormente, y como *próximo* el nodo que se visitará a continuación. Estudie el lado derecho de la figura de ejemplo. Programe la función *asignarPrev* que asigna los campos *prev* y *prox* agregados a la estructura de los nodos de un árbol *t*. El encabezado de la función se muestra a la derecha. El parámetro **pprev* es de entrada y salida. El nodo previo del primer nodo visitado (el nodo 1 en el ejemplo) debe ser el nodo apuntado inicialmente por **pprev* (nodo 0) y el nodo próximo del último nodo en ser visitado (nodo 5) debe ser NULL. En **pprev* debe quedar finalmente la dirección del último nodo visitado (nodo 5). En el siguiente ejemplo de uso las variables *t* y *prev* son de tipo Nodo *.

```
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
  struct nodo *prev, *prox;
} Nodo;
void asignarPrev(Nodo t,
                 Nodo **pprev);
```



Restricción: Su solución debe tomar tiempo linealmente proporcional al número de nodos en el árbol *t*.

Ayuda: Cuando visite el nodo T, su nodo previo es **pprev*. Asigne NULL a su nodo próximo por ahora. Si el nodo previo a T no es NULL, T es el nodo próximo del nodo previo a T. Antes de continuar el recorrido, asigne T a **pprev*.

Pregunta 4

La función *guardarItemes* de más abajo escribe un arreglo *items* de tamaño *n* en un archivo de nombre *archNom*. Programe la función *leerItemes* que retorna un arreglo de ítemes igual al que escribió *guardarItemes*, leído del archivo *nomArch* y entregando en **pn* el tamaño del arreglo leído. El encabezado de la función es: *Items *****leerItemes****(char *nomArch, int *pn);*

```
typedef struct {
  char *nom;
  double peso;
} Item;

void guardarItemes(
     char *nomArch,
     Item **itemes, int n) {
  FILE *arch= fopen(nomArch,
                    "w");
  fwrite(&n, sizeof(n),
         1, arch);
  for (int i= 0; i<n; i++) {
    Item *item= itemes[i];
    int len= strlen(item->nom);
    fwrite(&len,sizeof(len),
           1, arch);
    fwrite(item->nom, len,
           1, arch);
    fwrite(&item->peso,
           sizeof(item->peso),
           1, arch);
  }
  fclose(arch);
}
```

Al recrear el arreglo de ítemes, Ud. necesita pedir espacio con *malloc* para: el arreglo de ítemes, los *n* ítemes del arreglo y los *n* nombres almacenados en los ítemes.