CC3301 Programación de Software de Sistemas – Control 1
Semestre Primavera 2017 – Prof.: Luis Mateu

Pregunta 1 (60%)

Parte a.- (2 puntos) Programe la función *concat_bits* declarada como:

```
typedef unsigned int uint; // enteros sin signo
uint concat_bits(uint x, int n, uint y, int m);
```

Si los bits de x son $x_{31}\ x_{30}\ ...\ x_1\ x_0$ en donde x_0 es el bit menos significativo y los de y son $y_{31}\ y_{30}\ ...\ y_1\ y_0$, la función *concat_bits* debe retornar 0 … 0 $x_{n-1}\ ...\ x_0\ y_{m-1}\ ...\ y_0$. Considere que n y m son menores que 32 y $n+m<=32$. Ejemplos de uso:

```
uint z1= concat_bits(0b101, 3, 0b11001, 5);    // z1=0b10111001
uint z2= concat_bits(0b11011, 3, 0x10101, 2);  // z2=0x01101
```

Restricción: Ud. no puede usar los operadores de multiplicación, división o módulo (* / %). Use los operadores de bits eficientemente.

Parte b.- (4 puntos) Programe la función:

```
void insertar(char *d, char *s);
```

Esta función debe insertar el string *s* al principio del string *d*. Por lo tanto el string *d* cambia, mientras que *s* no cambia. Ejemplo de uso:

```
char d[80]= "perro"; // d es un arreglo de 80 bytes con el string "perro"
insertar(d, "gato"); // d es el string "gatoperro"
```

En el ejemplo *d* corresponde a un arreglo de 80 bytes, pero solo se ocupan los primeros 6 bytes para almacenar el string `"perro"` (no olvide el término del string). En el arreglo hay espacio más que suficiente para insertar al principio el string `"gato"`.

Restricciones: Ud. no puede usar el operador de subindicación [ ], ni su equivalente **(p+i)*. Para recorrer el string use aritmética de punteros. No puede pedir memoria adicional ni declarar otros arreglos de caracteres. Use múltiples punteros para direccionar distintas partes del string.

Sea cuidadoso al desplazar el contenido original de *d* para hacer espacio para *s*. Si copia ascendentemente, el resultado final de *d* en el ejemplo será incorrectamente *"gatoperrp"*. Hágalo descendentemente.

Pregunta 2 (40%)

Programe la función:

```
void podar(Nodo **pa, int y);
```

Esta función debe modificar el árbol de búsqueda binaria **pa* eliminando todos los nodos etiquetados con valores mayores que *y*. No necesita liberar la memoria de los nodos eliminados. Estudie los 3 ejemplos de uso de la figura de la derecha.

Restricciones: Sea eficiente, el tiempo de ejecución debe ser proporcional a la altura del árbol en el peor caso. No puede usar ciclos (como *while* o *for*). Debe usar recursión. No puede usar *malloc*.