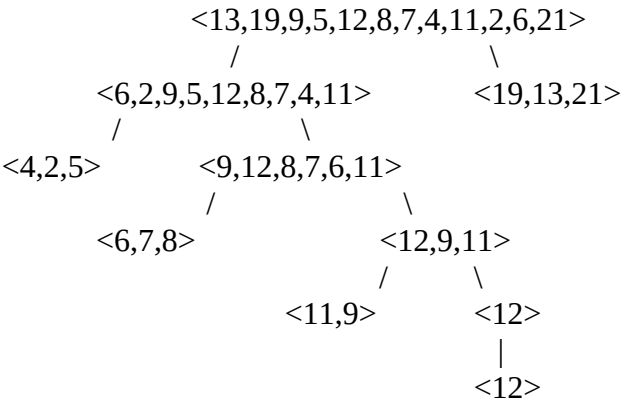
ELO320
Segundo Certamen 23/05/2001

TODAS LAS PREGUNTAS VALEN 20 puntos => total 100 puntos
90 minutos. Las respuestas estarán publicadas en la página del curso al término del certamen.

1.- Haga un árbol binario en que se muestre cada una de las particiones generadas por Randomized_Select(A, 1, 12, 9) al ser aplicado al arreglo A=<13,19,9,5,12,8,7,4,11,2,6,21>. Indique además cual es el valor retornado. Asuma que la función random usada en el algoritmo siempre retorna 1. Los índices del arreglo A van de 1 a 12.



Retorna 12.

2.- Implemente una cola usando tipos de datos dinámicos; es decir no arreglos. Como condición sus nodos solo se pueden enlazar en forma simple; es decir se debe usar una estructura de nodos simplemente enlazados. Las operaciones Enqueue y Dequeue deberían tomar tiempo O(1). Se pide:

a) La estructura de datos usada, y
b) La implementación de las operaciones Enqueue y Dequeue sobre esa estructura de datos.

Hay varias soluciones, una es la presentada aquí.

a)

```
typedef struct nodo_tag {
        struct cola_tag * next;
        elementType element;
} NODO_COLA;
typedef struct cola_tag {
        NODO_COLA * oldest;  /* puntero al nodo el más antiguo en la cola */
        NODO_COLA * youngest;  /* puntero al nodo el más joven en la cola */
} COLA;
```

b)

```
void Enqueue (COLA * C, NODO_COLA * e) {
        if (C->youngest == NULL)  /* se trata del primero que encolamos */
                C->youngest = C->oldest = e;
        else {
                C->youngest->next = e;
                C->youngest=e;
```

```
        }
}
NODO_COLA * Dequeue (COLA *C) {
        NODO_COLA * aux =C->oldest;
        assert (C->oldest != NULL); /* previene underflow */
        if (C->oldest == C->youngest)    /* desencolando el último */
                C->oldest =C->youngest = NULL;
        else
                C->oldest = C->oldest->next;
        return (aux);
}
```

3.- Dado n enteros en el rango 0 a k-1.

i) Codifique en C un algoritmo que procese la entrada - int [] A (OJO Aquí debió ser int A[], "mia culpa"), int n - en tiempo O(n+k), y posteriormente permita responder, en tiempo O(1), la consulta ¿cuántos de los n enteros caen en el intervalo (a,b]? $0 \leq a < b \leq k-1$.

ii) Codifique en C la función que responde a la consulta.

La idea es preprocesar en tiempo O(n+k) la entrada para después poder responder muy rápidamente a muchas consultas con distintos a y b.

Hay varias formas de codificarlo, una es la dada aquí.

i)

```
void Preprocesamiento(int A[], int n, int C[], int k) {  /* en C dejo valores procesados */
        int i;
        for (i = 0; i < k; i++)
                C[i]=0;
        for (i=0; i < n; i++)
                C[A[i]]++;
        for (i=1; i < k; i++)
                C[i]+=C[i-1];
}
```

ii)

```
int enRango(int C[], int k, int a, int b) {
        return (C[b]-C[a]);
}
```

4.- Proponga un algoritmo para determinar si una secuencia de paréntesis '(' , ')', '{' , y '}' está balanceada. Suponga que la secuencia es ingresada en un arreglo, A, de caracteres y de largo n.

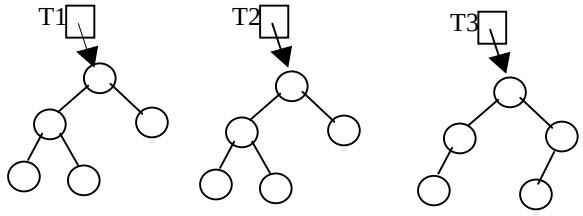
Entregue su respuesta como una función en C con parámetros A (un arreglo de caracteres) y n (el largo del arreglo). Ésta retorna 1 (verdadero) si la secuencia está balanceada y 0 si no lo está.

Por ejemplo: ({(){}}) está balanceada, pero ({(){)}) no lo está.

En su respuesta, usted puede hacer uso de todas las estructuras de datos y operaciones vistas en clases. Considere que almacenan caracteres como datos o claves. No necesita reescribir el código de estas operaciones.

```
Int EsSecuenciaBalanceada(char A[], int n) {
        int i;
        STACK S;
        MakeNull(S);
        for (i=0; i<n; i++) {
                switch (A[i]) {
                case '(', '{': Push(S, A[i]);
                        break;
                case ')', '}': if (Stack_Empty(S))
                                return (0);
                        if (Top(S) =='(' && A[i]=='}' || Top(S)=='{' && A[i]==')')
                                return (0);
                        Pop(S);
                        break;
                }
        }
        return(Stack_Empty(S));
}
```

5.- Escriba en C un algoritmo para determinar si dos árboles binarios tienen igual forma (no importa su contenido). Use la estructura dada en clases para representar cada nodo. Por ejemplo: T1 es igual en forma que T2, pero T1 y T2 no son iguales en forma a T3.




```
int SonIgualesEnForma(TREE_NODE * T1, TREE_NODE * T2) {
        if (T1==NULL && T2==NULL)
                return (1);
        if (T1 ==NULL && T2 != NULL)
                return(0);
        if (T1 != NULL && T2 == NULL)
                return(0);
        return (SonIgualesEnForma(T1->left,T2->left) &&
                SonIgualesEnForma(T1->right, T2->right));
}
```