ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

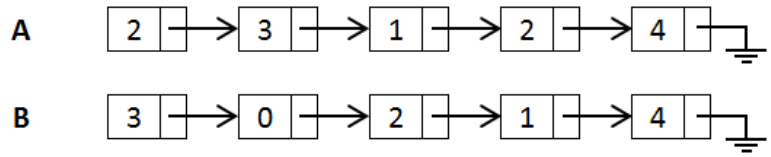
FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN
ESTRUCTURAS DE DATOS
PRIMERA EVALUACIÓN – II TÉRMINO 2012-2013

***Nombre:*_____*Matrícula:*_____*Paralelo:*___**

TEMA 1 (25 ptos)

Implemente la función **`List *SumaProducto(List *A, List *B);`** que recibe dos listas ***A*** y ***B***, con las cuales realiza operaciones de producto y suma, cuyos resultados van a formar parte de una tercera lista a retornar.

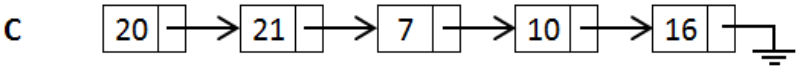
Ejemplo: Sean las listas A y B



Operaciones a realizar:

Nodo 1:	2*3	+	2*0	+	2*2	+	2*1	+	2*4	=	**20**
Nodo 2:			3*0	+	3*2	+	3*1	+	3*4	=	**21**
Nodo 3:					1*2	+	1*1	+	1*4	=	**7**
Nodo 4:							2*1	+	2*4	=	**10**
Nodo 5:									4*4	=	**16**

Resultado de SumaProducto:



NOTA: Asuma que A y B son de iguales dimensiones.

TEMA 2 (30 ptos)

El término "blog" fue acuñado por Jorn Barger en 1997. Un blog también conocido como weblog o bitácora, es un sitio web que recopila cronológicamente artículos de uno o varios autores, apareciendo primero el más reciente.
Habitualmente, en cada artículo, los lectores pueden escribir sus comentarios y el autor darles respuesta, de forma que es posible establecer un diálogo.
Para facilitar la búsqueda de artículos en los motores de búsqueda (Google, Yahoo, Altavista, etc.) se suelen agregar un conjunto de palabras claves a cada articulo, estas palabras claves van en función del contenido, representando o resumiendo en una o dos palabras de que se trata el articulo. Por ejemplo si un articulo aborda el tema de *Cómo resolver problemas usando estructuras de datos*, las palabras claves podrían ser: "TDA", "abstracción".

Defina usando lenguaje C, la o las TDA's necesarias para representar el problema anterior.

TEMA 3 (25 ptos)

Sea la TDA:

```
typedef struct {
        List *l_ordenada; Orden orden;
}PriorityQueue;
```

Donde *Orden* es un tipo de dato enumeración cuyos valores son ***ASC*** y ***DES***, que representa el tipo de ordenamiento de los elementos en la cola, y ***l_ordenada*** representa una lista ordenada.
Implemente el comportamiento:

```
void priorityQueueEnQueue(PriorityQueue *Q, NodeList *newNode, fnCmp compare)
```

El cual inserta un nuevo *nodo* en la ubicación que le corresponde de acuerdo al tipo de ordenamiento de la cola. El parámetro ***compare*** representa una función de *CallBack* que recibe dos parámetros que son los contenidos de los nodos a comparar y retorna ***0*** si son iguales, ***-1*** si el primero es menor que el segundo y ***1*** si el primero es mayor que el segundo.

TEMA 4 (20 ptos)

a) Qué imprime el siguiente bloque de código:

```
for(i=1;i<=10;i++)
   stackPush(st,nodelistNew(integerNew(i%6)));
while(!stackIsEmpty(st)){
   stackPush(s,stackPop(st));
   if(*(int*)nodelistGetCont(stackPeek(s))%2)
      stackPush(s,nodelistNew(integerNew(*(int*)nodelistGetCont(stackPop(s))*2)));
}
while(!stackIsEmpty(s))
   printf("%d ",integerGet(nodelistGetCont(stackPop(s))));
```

b) ¿Cuáles son las posibles salidas del código a continuación?

```
Int N=3;
Stack *p=stackNew();
for (i=1;i<=N;i++)
   if (TEST(i))
       printf("%d",i);
   else stackPush(p, nodelistNew(integerNew(i)));
while (!stackIsEmpty(p)){
   printf("%d", integerGet(nodelistGetCont(stackPop(p))));
}
```

a) 1 2 3
b) 1 3 2
c) 2 1 3
d) 2 3 1
e) 3 2 1
f) 3 1 2

Nota: La función TEST genera valores de 0 y 1 de forma aleatoria.

Referencia

generic.h

```
typedef void * Generic;
typedef Generic (*readfn)(FILE *F);
typedef int (*cmpfn)(Generic, Generic);
typedef void (*printfn) (Generic);
Generic integerNew(int newValue);
void integerSet(Generic g, int newValue);
int integerGet(Generic g);
Generic charNew(char newValue);
void charSet(Generic g, char newValue);
char charGet(Generic g);
```

nodelist.h

```
NodeList *nodeListNew(Generic g);
void nodeListSetCont(NodeList *p, Generic g);
Generic nodeListGetCont(NodeList *p);
void nodeListSetNext(NodeList *p, NodeList *q);
NodeList *nodeListGetNext(NodeList *p);
int nodeListHasNext(NodeList *p);
```

list.h

```
List *listNew();
int listIsEmpty(List *L);
NodeList *listGetHeader(List *L);
NodeList *listGetLast(List *L);
void listAddNode(List *L, NodeList *newNode);
void listAddFirst(List *L, NodeList *newNode);
void listAddNext(List *L, NodeList *p, NodeList *newNode);
NodeList *listSearch(List *L, Generic value,cmpfn f);
NodeList *listGetPrevious(List *L, NodeList *p);
void listRemoveNode(List *L, NodeList *p);
NodeList *listRemoveFirst(List *L);
NodeList *listRemoveLast(List *L);
int listNodeExists(List *L, NodeList *p);
int listGetSize(List *L);
void listPrint(List *L, printfn print);
List *listReadFile(char *fileName, readfn read);
```

stack.h

```
int stackIsEmpty(Stack *S);
Stack * stackNew();
void stackPush(Stack *S, NodeList *nuevo);
NodeList *stackPop(Stack *S);
void stackDelete(Stack **S);
NodeList *stackPeek(Stack *S);
int stackGetSize(Stack *S);
```

queue.h

```
int queueIsEmpty(Queue *St);
Queue *queueNew();
void queueEnQueue(Queue *St, NodeList *nuevo);
NodeList *queueDeQueue(Queue *St);
void queueDelete(Queue **St);
NodeList *queueGetFront(Queue *St);
NodeList *queueGetLast(Queue *Q);
int queueSize(Queue *Q);
```