Estructuras de Datos y Algoritmos

EXAMEN – 21/12/2009

Nombre y Apellido	C.I.	Turno

Por favor siga las siguientes indicaciones:

- Escriba con lápiz.
- Escriba su nombre y número de documento en todas las hojas que entregue.
- Numere las hojas e indique el total de hojas en la primera de ellas.
- Escriba las hojas **de un solo lado**.
- Comience cada ejercicio en una hoja nueva.
- El total máximo de puntos es **100**.
- El examen contiene un total de: **10** páginas.
- La prueba es individual y sin material.
- Solo se contestan dudas acerca de la letra de los ejercicios.

	Problema 1	Problema 2	Problema 3	Total:
Solo para uso docente				

Problema 1 (44 puntos)

Parte a) (8 ptos.)

De una especificación mínima funcional del tipo abstracto de dato Queue (Cola). Comente brevemente y de forma clara las funciones especificadas.

Parte b) (8 ptos.)

De una especificación mínima del tipo abstracto de dato Stack (Pila). Las operaciones que afectan el Stack deben implementarse como procedimientos que modifican el parámetro de entrada. Dicho de otro modo: no deben ser funcionales. Comente brevemente y de forma clara las operaciones especificadas.

Parte c) (28 ptos.)

Utilizando los TADs especificados en a) y b), escriba un procedimiento iterativo que reciba una secuencia de caracteres en una Queue (Cola) y la procese de la siguiente manera:

La secuencia puede contener 2 tipos de paréntesis, las parejas: "(*", "*)" y "[", "]". Las mismas encierran comentarios que deben ser eliminados de la cadena (Queue) de entrada. éstos pueden estar anidados uno dentro de otro, pero respetando las siguientes condiciones:

toda marca de apertura, "[" o "(*", debe tener su correspondiente marca de cierre y cada marca de cierre, "]" o "*)", se aparea con la última de apertura

El procedimiento debe eliminar de la tira de caracteres todos los comentarios, así como verificar si la secuencia es correcta.

La tira de caracteres se recibe en una Queue con un car´cter especial que indica el fin de la secuencia, el mismo es "$". No se puede utilizar ninguna cola auxiliar, es decir, se debe devolver la tira sin los comentarios en la misma Queue que se recibe. En caso de que la tira contenga errores, se debe retornar **true** en el parámetro booleano de salida error (en este caso no importa lo que se devuelva en la tira), en otro caso error debe valer **false**.

El cabezal del procedimiento es el siguiente:

void PreProcesador (Queue &tira, boolean & error);

Ejemplos:

Entrada Salida tira Salida error
Procedure p; (* calcula perímetro*) q := 0$ Procedure p; q := 0$ FALSE
Este es [me(*no*)s] original$ Este es original$ FALSE
Aca [se (*muestran 2] niveles incompletos$ TRUE
Otro co[n err(*or p]or es*)tar intercalado$ TRUE

Nota: el tipo de los elementos de la pila y la cola es char.

SOLUCIONES
Parte a)

```
typedef struct NodoCola* Cola;

Cola NullQ();
/* Crea la cola vacia */
Cola EnQueue( CHAR c,  Cola q);
/* Dado un caracter y una cola devuelve la cola resultado de haber insertado el
caracter al final de la cola q */
bool EmptyQ(Cola q);
/* Retorna TRUE sii la cola es vacia */
char Front(Cola q) ;
/* Pre: La cola q no es vacia. Devuelve el elemento que esta en el frente de la cola */
Cola DeQueue(Cola q ) ;
/* Pre: La cola q no es vacia. Devuelve la cola q sin el elemento que esta en el frente
de la cola */
```

Parte b)

```
typedef  NodoStack* Pila;

void NullS(PilaS&);
/* Crea la pila vacia */
void Push(CHAR c, Pila & s);
/* Dado un caracter y una pila devuelve la pila resultado de haber insertado el
caracter en el tope de la pila s */
bool EmptyS(Pila s) ;
/* Retorna TRUE sii la pila es vacia */
void Top(Pila s) ;
/* Pre: La pila s no es vacia. Devuelve el elemento que esta en el tope de la pila */
void Pop(Pila &s);
/* Pre: La pila s no es vacia. Devuelve la pila s sin el elemento que esta en el tope de
la pila */
```

Parte c)

La tira por lo menos tiene el centinela. Utilizamos los TADs de las partes anteriores. Utilizaremos una variable 'coment' de tipo Boolean. Esta determina si estoy en un comentario o si es texto valido para devolver.

El algoritmo utilizara un stack de comentarios. Cada vez que comienza un comentario, hacemos un Push sobre el stack y establece la variable coment en estado TRUE. Si el comentario es de tipo "[" entonces : Push('[',s), en cambio si es un "(*", entonces : Push('(',s). Ver que no es necesario hacer un Push de '(' y luego un Push de '*' en el stack ya que tengo que solo diferenciar entre los dos tipos de comentarios, esto lo puedo hacer solo guardando un '[' o un '(' en el stack.

Si viene un cierre de comentario, entonces se controla con lo que hay en el tope del stack (sino esta vació). Si concuerda, en cuanto al tipo de comentario, entonces Pop sobre el stack y se controla si esta vació para cambiar el estado de la variable coment a FALSE.

Al final se controla que se hayan cerrados todos los comentarios correctamente.

```
#define TRUE true
#define FALSE false


void PreProcesador (Cola& tira ,bool &error);
const char cent = '$';
//declaración de variables
 Pila s;
 bool coment;

        /* Inicializar variables */
        coment = FALSE;
        NullS(s);
        error = FALSE;
        while((!error) && (Front(tira) != cent) ){
                /* Evaluar que hay en el frente de la tira */
                switch (Front(tira)){
                case '[' :/* Empieza comentario */
                                Push(Front(tira),s); coment = TRUE; break;
                case '(' :/* Puede empezar comentario */
                                if (Front(DeQueue(tira)) == '*') { /* Comienza
comentario ? */
                                        tira = DeQueue(tira);
                                        Push('(',s); coment := TRUE;
                                }ELSE{
                                        /* No era comentario, entonces devuelvo si no
estoy en zona de comentarios */
                                        IF (! coment)  /* Devuelvo si corresponde */
                                                tira := EnQueue(Front(tira),tira);

                                };break;
                case ']' :/* Cierra comentario */
                                IF (! EmptyS(s)) {
                                        IF (Top(s) == '[') {/* Corresponde este tipo ? */
                                                Pop(s);
```

```
                                        IF (EmptyS(s) ) /* Si el stack es vacio,
entonces empieza texto valido */
                                                coment = FALSE;
                                    else
                                        /* No correspondia este tipo de cierre,
entonces ERROR */
                                        error = TRUE;
                                    }
                                else
                                    /* No correspondia cerrar dado que no habia
nada en el stack, entonces ERROR */
                                    error = TRUE;
                                } break;
                        Case '*' : /* Puede ser un cierre de comentario */
                                IF (Front(DeQueue(tira)) == ')') {/* Cierre de comentario
? */
                                    tira:= DeQueue(tira);
                                    if (! EmptyS(s)) {
                                        if (Top(s) == '(') { /* Corresponde este tipo
? */
                                            Pop(s);
                                            if (Empty(s)) {/* Si el stack es
vacio, entonces empieza texto valido */
                                                coment = FALSE;
                                            }
                                        }else{
                                            /* No correspondia este tipo de
cierre, entonces ERROR */
                                            error = TRUE;
                                        }
                                    }else{
                                        /* No correspondia cerrar dado que no
habia nada en el stack, entonces ERROR */
                                        error = TRUE;
                                    }
                                }else{
                                    /* No era comentario, entonces devuelvo si no
estoy en zona de comentarios */
                                    if (! coment)
                                        tira = EnQueue(Front(tira),tira);

                                }
                        }else{
                            /* Si estoy en zona de comentarios devuelvo, sino no hacer
nada */
                            if (! coment)
                                tira = EnQueue(Front(tira),tira);

                        }
                        /* Avanzar en la tira */
```

```
                tira = DeQueue(tira);
        }

        if(! error) {
                If (EmptyS(s) ){/* Quedaron todos los comentarios cerrados ? */
                        /* Si esta todo ok, entonces poner centinela al final */
                        tira = EnQueue(Front(tira),tira);
                        tira = DeQueue(tira);
                }else{
                        /* Algun comentario quedo sin cerrar */
                        error = TRUE;
                }
        }
}
```

Problema 2 (16 puntos)

Sea **A** la función de Ackermann tal que:

A(m,n)=

- n+1 si m=0
- A(m-1,1) si m>0 y n=0
- A(m-1,A(m,n-1)) si m>0 y n>0
- n,m pertenecen a N

Implemente en C/C++ un único procedimiento que implemente la función de Ackermann, que reciba dos argumentos que representan m y n, y devuelva en un tercer argumento el resultado de la función.

Problema 3 (40 puntos)

Parte a) (2 puntos)

De una representación para el tipo **árboles binarios** de enteros, que permita que la estructura de árbol pueda crecer de forma indefinida.
Nombre al tipo de su representación cómo ABZ es decir:

```
...
typedef struct ABZ{
...
...
};
```

Solución:

```
typedef struct ABZ{
      int raiz;
      ABZ* izq,*der;
};
```

Parte b) (15 puntos)
Implemente la función **bool esSubArbol(ABZ* a,ABZ *b)**

Que recibe como parámetros dos **árboles binarios** de enteros y retorna verdadero si el árbol b es subárbol de a, y retorna verdadero si b es subárbol de a
Tener en cuenta que la comparación de los elementos se realizara a nivel del valor de los nodos

Solución:

```
bool esSubArbol(ABZ* a,ABZ* b){
```

```
 if(b==NULL) return true;
 else{
   if(a==NULL)return false;
   else{
     if(a->raiz==b->raiz){
          bool essubizq= minesSubArbol(a->izq,b->izq);
          bool essubder= minesSubArbol(a->der,b->der);
         if(essubizq&&essubder) return true;
         else{
               //sino sigo buscando
                return esSubArbol(a->izq,b) || esSubArbol(a->der,b);

         }


      }else{
          //sino sigo buscando
           return esSubArbol(a->izq,b) || esSubArbol(a->der,b);
      }

   }

 }
}


bool  minesSubArbol(ABZ* a,ABZ* b){
 if(b==NULL) return true;
 else{
   if(a==NULL)return false;
   else{
     if(a->raiz==b->raiz){
          bool essubizq= minesSubArbol(a->izq,b->izq);
          bool essubder= minesSubArbol(a->der,b->der);
         if(essubizq&&essubder) return true;
         else{
                        return false;
         }


      }else{
           return esSubArbol(a->izq,b) || esSubArbol(a->der,b);
      }

   }

 }
}
```

Parte c) (10 puntos)
Implemente el procedimiento **void obtenerSimetrico(ABZ* a, ABZ*& b)**

Que devuelve en el parámetro b el simétrico del **árbol binario** a, la variable b no comparte memoria con a.

Solución:

```
void obtenerSimetrico(ABZ* a, ABZ*& b){
        if(a!=NULL){
        b=new ABBZ;
        b->raiz=a->raiz;
        obtenerSimetrico(a->izq,b->der);
        obtenerSimetrico(a->der,b->izq);
       }else b=NULL;


}
```

Parte d) (13 puntos)
Implemente la **función int borrarElemento(ABZ*& a, int n)**
Que borra todas los nodos (baja física) que contienen el valor n, en el **árbol binario** a y retorna la cantidad de elementos borrados.

La estrategia elegida para borrar elementos es la siguiente:

1-Si el nodo a borrar no tiene hijos se da de baja.
2-Si el nodo a borrar tiene un hijo izquierdo, se sustituye el valor del nodo a borrar por el valor de un nodo hoja de su subárbol izquierdo, y se borra físicamente esta hoja.
3-Si el nodo a borrar tiene solo un hijo derecho, se sustituye el valor del nodo a borrar por una hoja de su subárbol derecho, y se borra físicamente esta hoja.

En todas las secciones del ejercicio 3, se pueden utilizar funciones y procedimientos auxiliares, pero en caso de hacerlo debe implementarlos.
El tipo de estructura manejada es árbol binario por lo que los elementos no se encuentran distribuidos en algún orden determinado. Además en todos los ejercicios se permite que la estructura tenga elementos repetidos.

Solución:

```
int borrarElemento(ABBZ*& a, int n){
     if(a==NULL) return 0;
```

```
    else{
            int cont=0;
            if(a->raiz==n){
             cont++;
             if(a->izq!=NULL || a->der!=NULL){
               a->raiz=borrarNodoHoja(a);
               cont=borrarElemento(a->izq,n)+borrarElemento(a->der,n);


             }else{
                  delete a;a=NULL;


            }

            }



    }
    return cont;
}


int borrarNodoHoja(ABBZ*&a){
        //Borra físicamente una hoja de a y devuelve su valor
       //en caso de que el arbol tenga solo un nodo, devuelve el valor entero que
//contiene y lo borra fisicamente

if(a!=NULL){
      if(a->der==NULL && a->izq==NULL){
            int ret=a->raiz;
            delete a;
            a=NULL;
            return ret;
       }
      //else
      If(a->izq!=NULL){
            return(borrarNodoHoja(a->izq));
       }else{
         return(borrarNodoHoja(a->der));

      }
}else return 0;

}
```