


Apellido, Nombre: Calificación: 5 (cuer)

Serán considerados al calificar este examen la eficiencia de las soluciones y la utilización adecuada de las características del lenguaje C y de la programación estructurada.

Para aprobar es necesario obtener al menos 5 puntos en este examen.

Al menos el 30% de cada uno de los ejercicios 2, 3 y 4. Al menos 4,25 entre los ejercicios 2, 3 y 4

Cuando este examen está aprobado, la nota FINAL se obtiene así: $CURSADA * 0.3 + TOTALIZADOR * 0.7$

En todos los ejercicios que corresponda, mostrar las invocaciones (incluyendo su contexto: declaraciones, inicializaciones y acciones posteriores) de las soluciones desarrolladas.

(1,5p) Ej 1.- Indicar Verdadero o Falso, justificando o ejemplificando adecuadamente (de lo contrario tendrá puntaje cero)

a) La implementación circular de una Cola, permite aprovechar los espacios libres que van quedando al poner y quitar elementos de ésta.

b) Si luego de una rotación simple sobre un árbol AVL que se había desbalanceado, el mismo sigue estando desbalanceado implica que debió haberse aplicado una rotación compuesta.

(4p) Ej 2.- En un torneo de tenis se venden entradas para cada partido y tales ventas se gestionan mediante una lista simplemente enlazada donde cada nodo contiene:

- CodPartido (cadena de 6) ordenada por este criterio
- CapacidadEstadio
- CantidadEntradasVendidas
- Sublista Entradas Vendidas, donde cada elemento contiene:
 - Apellido y nombre -ANU25- del titular de la compra
 - Cant de entradas compradas
 - Mail de contacto
 - Tipo Entrada -char- (A/B/C) ordenada por este criterio

Definir tipos de la lista y resolver usando TDA Cola:

a) Se ha recibido el archivo binario COMPRAS.DAT que posee los datos necesarios para una compra.

i) Definir el tipo registro del archivo COMPRAS.DAT con los datos que considere necesarios para efectuar la compra.

ii) Procesar el archivo COMPRAS.DAT almacenando cada compra que se pueda realizar en la sublista del nodo correspondiente. Para esto considerar que puede haber en el archivo datos de Partidos erróneos y que el aforo para cada partido será del 40%, superado ese valor, la venta no podrá realizarse. Informar la situación y almacenar los datos de las ventas no realizadas por superarse el aforo del Estadio, en una Cola que contenga Apellido y Nombre, Mail y CodPartido.

b) Definir el tipo de la Cola y desarrollar los operadores utilizados en la solución de ii), suponiendo la misma implementada estáticamente.

(2,5p) Ej 3.- (Utilizar TDA N-Ario) Se tiene un árbol binario AB proveniente de la transformación de un bosque y un árbol N-ario AN; ambos de enteros. Hallar e informar si se verifica que la suma de las claves de cada uno de los árboles del bosque se encuentra en alguna hoja de AN.

(2p) Ej 4.- Dado un grafo conexo con aristas ponderadas implementado en una lista de adyacencia en la cual en cada nodo de la lista se han marcado con 1 las aristas que son parte del AAM (0 en caso contrario). Hallar e informar: el costo del AAM, la arista de mayor peso del AAM y los vértices que une (si hubiera mas de una con el peso mayor, mostrar la última encontrada). Definir tipos.

Julio 2021

1 - V o F

a) La implementación circular de una cola permite aprovechar los espacios libres que van quedando al poner y quitar los elementos de esta VERDADERO. En la cola circular se aprovechan los lugares libres

2- Torneo de tenis, se venden entradas para cada partido

Lista SE {
 CodPartido
 CapaEstadio
 CantEntVendidas
 Sublista de Ent. vendidas } {
 Apellido y nombre
 Cant de Entradas compradas
 Mail de contacto
 Tipo entrada

a) Archivo binario COMPRAS.DAT (datos necesarios para una compra)

i) Definir el tipo de registro del archivo con los datos nec. para la compra

main.c

```
#include <stdio.h> #include <stdlib.h> #include <string.h> #include "TDACola.h"
```

```
#define COD 7 #define ANU 26 #define COEE 50
```

```
typedef struct {  
  char CodPartido[COD], Titular[ANU], Mail[COEE], Tipo;  
  unsigned short int Pedidas; } TregCompra;
```

```
typedef struct nodito {  
  char Titular[ANU], Mail[COEE], Tipo;  
  unsigned short int Compradas;  
  struct nodito *sig; } nodito;
```

```
typedef nodito *Tsub;
```

```
typedef struct nodo {  
  char CodPartido[COD];  
  unsigned int Capacidad, CantVendidas;  
  Tsub SubVendidas;  
  struct nodo *sig; } nodo;
```

```
typedef nodo *Tlista;
```


ii) Procesar el archivo, compra $\begin{cases} \text{realizada} \Rightarrow \text{Sublista correspondiente} \\ \text{no realizada} \Rightarrow \text{Cola} \end{cases}$

```
void ProcesaOrdnes(TLista LPartidos, TCola *Crechazadas) {
```

```
    Tlista PartidoAct;
```

```
    TSub ants, acts, nuevito;
```

```
    TelementoC Rech;
```

```
    FILE *archb; TregCompra RCompra;
```

```
    if ((archb = fopen("COMPRAS.DAT", "rb")) == NULL) {
```

```
        printf("No pudo abrirse el archivo");
```

```
    } else {
```

```
        while ((fread(&RCompra, sizeof(TregCompra), 1, archb)) == 1) {
```

```
            PartidoAct = LPartidos; //copiar datos en Rech;
```

```
            while (PartidoAct && strcmp(RCompra.CodPartido, PartidoAct->CodPartido) > 0) {
```

```
                PartidoAct = PartidoAct->sig;
```

```
            if (PartidoAct && strcmp(RCompra.CodPartido, PartidoAct->CodPartido) == 0) //Partido Valido
```

```
                if (PartidoAct->CantVendidas < 0.4 * PartidoAct.Capacidad) {
```

```
                    acts = PartidoAct->SubReservas;
```

```
                    while (acts != NULL && acts->Tipo < RCompra.Tipo) {
```

```
                        ants = acts;
```

```
                        acts = acts->sig;
```

```
                    }
```

```
                    nuevito = (nuevo *) malloc(sizeof(nuevo));
```

```
                    nuevito->Compradas = RCompra.Pedido;
```

```
                    strcpy(nuevito->Titular, RCompra.Titular);
```

```
                    strcpy(nuevito->Mail, RCompra.Mail);
```

```
                    nuevito->Tipo = RCompra.Tipo;
```

```
                    nuevito->sig = acts;
```

```
                    if (acts == PartidoAct->SubReservas)
```

```
                        PartidoAct->SubReservas = nuevito;
```

```
                    else
```

```
                        ants->sig = nuevito;
```

```
                }
```

```
            } else {
```

```
                printf("orden rechazada, exceso de capacidad %.s", RCompra.Cod);
```

```
                Ponec(Crechazados, Rech);
```

```
            }
```

```
        } else {
```

```
            printf("orden rechazada, partido invalido %.s", RCompra.Cod);
```

```
            Ponec(Crechazados, Rech);
```

```
        }
```

```
    }
```

```
    fclose(archb);
```

```
}
```

```
}
```

b) Definir el tipo de la cola y desarrollar los operadores utilizados

TDACola.h

```
#define ANU 26 #define COEE 50 #define COD 7 #define MAX 100
typedef struct {
    Char Nombre[ANU], Mail[COEE], CodPartido[COD]; } Telementoc;
typedef struct {
    Telementoc datos[MAX];
    int pri, ult; } TCola;
```

TDACola.c

```
#include <stdio.h> #include "TDACola.h"

int VaciaC(TCola c) {
    return c.pri == -1;
}

void Ponerc(TCola *c, Telementoc x) {
    if ((*c).ult != MAX - 1) {
        if ((*c).pri == -1)
            (*c).pri = 0;
        (*c).datos[++(*c).ult] = x;
    }
}

void SacarC(TCola *c, Telementoc *x) {
    if ((*c).pri != -1) {
        *x = (*c).datos[(*c).pri];
        if ((*c).pri == (*c).ult)
            IniciarC(c);
        else
            (*c).pri++;
    }
}

void IniciarC(TCola *c) {
    (*c).pri = -1;
    (*c).ult = -1;
}

Telementoc ConsultarC(TCola c) {
    if (c.pri != -1)
        return c.datos[c.pri];
}
```


→ Prov. de la trans. de un bosque

3- AB y AN ambos enteros. Informar si se verifica que la suma de las claves de cada uno de los arboles del bosque se encuentra en alguna hoja de AN

```
int sumaArbol(arbol AB) {
```

```
    if (AB != NULL)
```

```
        return AB->dato + sumaArbol(AB->izq) + sumaArbol(AB->der);
```

```
    else
```

```
        return 0;
```

```
} int EstaEsHoja(ArbolN AN, posicion P, int Clave) {
```

```
    if (!Nulo(P))
```

```
        if (Nulo(HijoMasIzq(P, A))
```

```
            if (Clave == Info(P, A))
```

```
                return 1;
```

```
            else
```

```
                return EstaEsHoja(AN, HnoDer(P, A), Clave);
```

```
        else
```

```
            return
```

```
                EstaEsHoja(AN, HijoMasIzq(P, A), Clave) || EstaEsHoja(AN, HnoDer(P, A), Clave);
```

```
    else
```

```
        return 0;
```

```
} int Verifica(arbol AB, ArbolN AN) {
```

```
    int Cumple = 1; suma; arbol a = AB
```

```
    while (AB != NULL && Cumple) {
```

```
        suma = AB->dato + sumaArbol(AB->izq);
```

```
        Cumple = EstaEsHoja(AN, Raiz(AN), suma);
```

```
        AB = AB->sig;
```

```
    }
```

```
    return Cumple && a != NULL;
```

```
}
```