Primer parcial – 9 de mayo de 2025

Nombre y Apellido: Valentina Acevedo Frías **Calificación:** Aprobado (P.P.)

IMPORTANTE: NO SE CORREGIRA LO QUE NO SEA LEGIBLE: PUEDE ENTREGAR EN LAPIZ SIEMPRE QUE SEA OSCURO
Serán considerados al calificar este examen la eficiencia y legibilidad de las soluciones y el uso de las características del lenguaje C y de la programación estructurada. Para facilitar el seguimiento del código, se sugiere numerar las llaves de cada bloque, o marcar con una línea las llaves asociadas.
*Para aprobar es necesario obtener **al menos 5p**, de los cuales **al menos 4,25p** deben obtenerse en el **inciso i)***
*Para acceder al coloquio de promoción, es necesario obtener **al menos 6p**, de los cuales **al menos 5p** deben obtenerse en el **inciso i)***

i) Inciso a) (3.5 p)	i) Inciso b) (5 p)	ii) y iii) (1,5p)	Calificación
3,50	3,50	1,35	8,35

Un supermercado registra la **compra** realizada por cada **cliente** en una **cola C** donde cada elemento representa un producto comprado y tiene los siguientes datos: *código de barra* (ANU 15), *cantidad comprada* (real).

Los productos que el supermercado ofrece a la venta se almacenan en una **lista simplemente enlazada LS**, un nodo por producto, con los siguientes campos: *código de barra* (ANU15, ordenado, no se repite), *precio unitario* de venta (real), *stock disponible* (real), *margen de ganancia* (porcentaje real mayor a 0), **sublista** de **compras** a **proveedores**, un nodo por compra: *fecha* (formato aaaa/mm/dd), *precio unitario* (real), *cantidad comprada* (real).

Una lista **doblemente enlazada LD** posee los productos con stock en negativo, en cada nodo: *código de barra* (ANU15, puede repetirse), *stock* (real)

Se pide desarrollar un programa en lenguaje C, modularizado y utilizando TDA cuando corresponda que:

i) resuelva:

a) Procesar la compra de **C**. Todos los productos de **C** existen en **LS**, en la cual debe actualizarse el stock disponible; cuando éste quede con un valor negativo debe agregarse al final de **LD**, *independientemente si antes de la actualización era positivo o ya estaba como negativo*. Al finalizar el proceso, mostrar el importe total de la compra y el importe total promedio por producto. No es necesario preservar los elementos de **C**.

b) En un archivo de texto PROVEEN.TXT se tiene en cada línea una **compra** realizada por el supermercado a un **proveedor**, con los siguientes datos: *fecha* (formato aaaa/mm/dd), *código de barra* (ANU15), *cantidad comprada* (real), *precio unitario compra* (real). El archivo no se encuentra ordenado por ningún criterio. Actualizar **LS** con la información del archivo, *contemplando solamente las compras de abril de 2025*, actualizando el stock y el precio unitario de venta (calcularlo recargando con el margen de ganancia al precio unitario de compra); contemplar que el producto comprado puede no existir en **LS** debiendo insertarlo, asignando al nuevo producto un 50% de margen de ganancia. Agregar la compra al principio de la sublista de compras a proveedores (puede estar vacía). Si luego de procesar la compra, el stock queda con valor positivo, se deben eliminar de **LD** las apariciones del producto comprado.

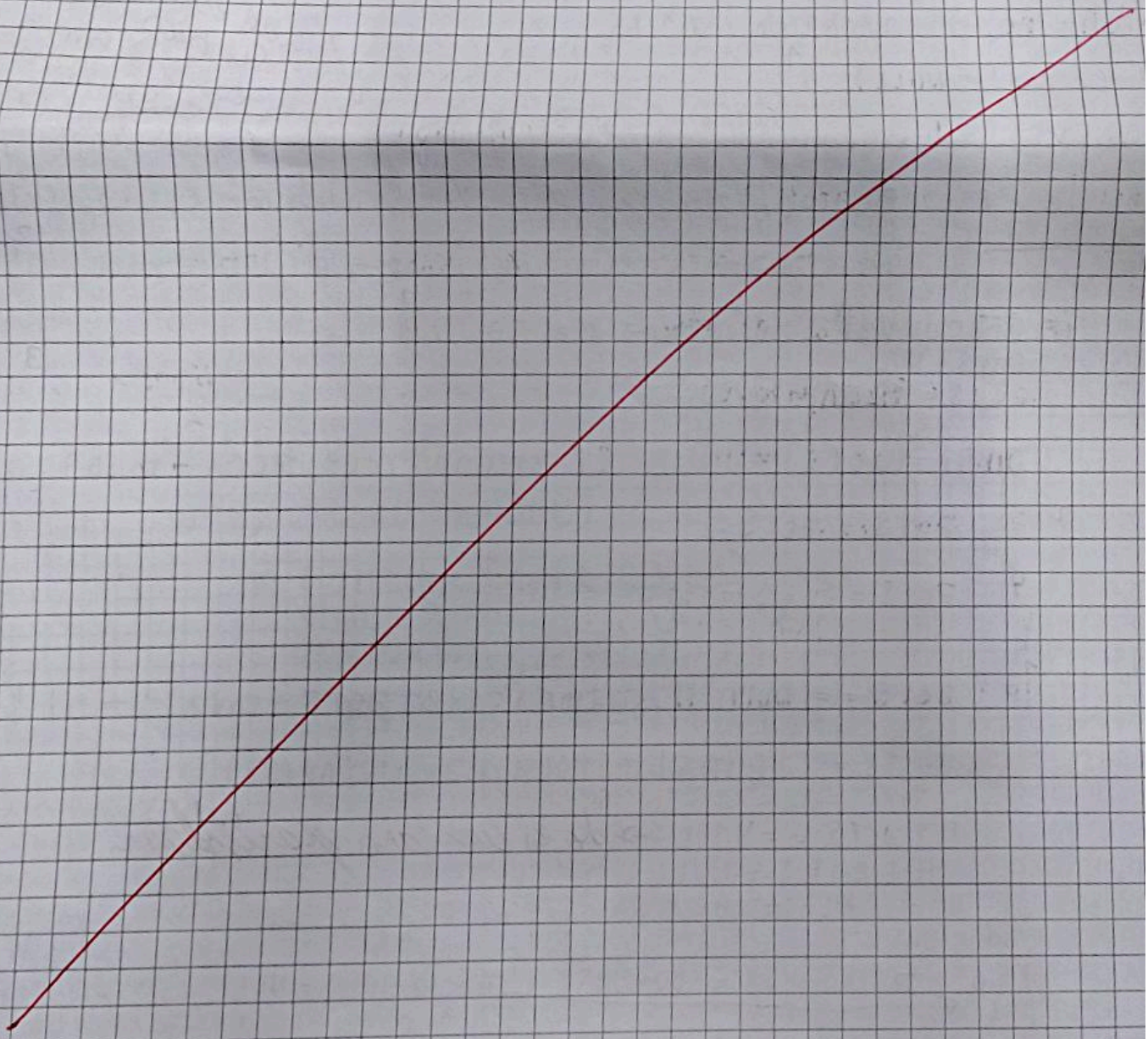
) incluya el *main()* completo que realice las invocaciones a los subprogramas definidos en **i)** además de las eclaraciones de tipos e inicialización de estructuras necesarias, e invocación a funciones de carga de las smas (que no debe desarrollar).

defina el tipo de la cola (implementación estática) utilizado en el programa y del elemento que contiene. sarrollar **SacaC() y VaciaC()**. Indicar en qué archivo/s iría cada definición/desarrollo.

(1) a)

```
void ejerA ( TListaS LS, TCola *C, TListaD *LD) {0}
  TElemC datoC;
  TListaS auxS;
  ProdoD nuevo;
  float importeTotal = 0, cantProd = 0;   (unsigned int)

  while (!Vacia (*C)) {1}
    Saca (C, &datoC);
    auxS = LS;
    while ( strcmp (datoC.codB, auxS->codB) > 0)
                auxS = auxS->sig;
    auxS->stock -= datoC.cantC;
    cantprod++;
    importeTotal += datoC.cantC * auxS->precioUV;
    IF (auxS->stock < 0) {3}
      nuevo = (ProdoD) malloc (sizeof (nodoD));
      strcpy (nuevo->codB, auxS->codB);
      nuevo->stock = auxS->stock;
      nuevo->sig = NULL;
      IF ((*LD).ult == NULL) {4}
        (*LD).ult = (*LD).pri = nuevo;
        nuevo->ant = NULL;
      {4}
      else {5}
        nuevo->ant = (*LD).ult;
        (*LD).ult->sig = nuevo;
        (*LD).ult = nuevo;
      {5}
  {4}
```

```
    }
}
printf("Importe total de la compra: %f", importTotal);
if (cantProd)
    printf("Importe total promedio por producto: %f",
           importTotal / cantProd);
}
```



b)

```
int fechaValid (char fecha[]) {
  return strcmp(fecha, "2025/04/01") >= 0 &&
                strcmp(fecha, "2025/04/30") <= 0;
}

void ejerB (char nombArch[], TListaS *LS, TListaD *LD) {   0
  FILE *arch;
  PnodoD auxD, elim;
  TListaS antS, actS, nuevoS;        sublista nuevaComp;
  char codB[COD];                    int eraNeg;

  arch= fopen (nombArch, "r");
  if (arch!=NULL) {   1
  nuevaComp=(sublista) malloc (sizeof (nodito));
    while (fscanf(arch, "%s %s %f %f", nuevaComp->fecha, codB,
               &nuevaComp->cantComp, &nuevaComp->precioU)==4)   2 {
      if (fechaValid (nuevaComp->fecha)) {   3
        antS = NULL;
        actS = *LS;
        while (actS!=NULL && strcmp(codB, actS->codB)>0) {   4
            antS = actS;
       4    actS = actS->sig;
        }

        if (actS == NULL || strcmp(codB, actS->codB)!=0) {   5
           nuevoS = (TListaS) malloc(sizeof(nodoS));
           if (actS == NULL)
             if (antS== NULL)
                *LS = nuevoS;
           else
         6     antS->sig = nuevoS;     nuevo->sig = NULL; // esta sentencia fuera
        }                                                  // del if
        else
```

→ esto es un caso particular

¿Cuál es el sentido de guardar sin saber si la compra es de abril?? (*)

```
if (actS == *LS) {
    nuevoS->sig = *LS;
    *LS = nuevoS;
}
else {
    antS->sig = nuevoS;
    nuevoS->sig = actS;
}
nuevoS->ganan = 50;    strcpy(nuevoS->codB, codB);
nuevoS->sub = NULL;
nuevoS->stock = 0;
actS = nuevoS;
}

eraneg = actS->stock < 0;
actS->stock += nuevocomp->contcomp;
actS->precioUV = (act->ganan/100+1) * nuevacomp->precioU;
nuevacomp->sig = actS->sub;
actS->sub = nuevacomp;
if (eraneg && actS->stock >= 0) {
    auxD = (*LD).pri;
    while (auxD != NULL) {
        if (strcmp(codB, auxD->codB) == 0) {
            elim = auxD;
            if (auxD == (*LD).pri) {
                (*LD).pri = (*LD).pri->sig;
                if ((*LD).pri == (*LD).ult)
                    (*LD).ult = NULL;
                else
                    (*LD).pri->ant = NULL;
                auxD = (*LD).pri;
            }
```

Corrections (margin notes):

- [illegible] instrucción!!
- El stock debía quedar con valor positivo
- ¡Modularizar!
- Ya movió pri!
- NULL

4/6

MODULARIZAR

```
                else
                    if (aux0 == (*LD).vit) {
                        (*LD).vit = (*LD).vit->ant;
                        (*LD).vit->sig = NULL;  aux0 = NULL;
                    }
                    else {
                        aux0->ant->sig = aux0->sig;
                        aux0->sig->ant = aux0->ant;
                        aux0 = aux0->sig;
                    }
                free (elim);
            }
            else aux0 = aux0->sig;
        }
    nueva comp= (sublista) malloc (sizeof (nodo));
    }
    Free( nueva comp);
    Fclose (arch);
    }
    else
        printf("no se pudo abrir el archivo");
}
```

duplicado en el .h

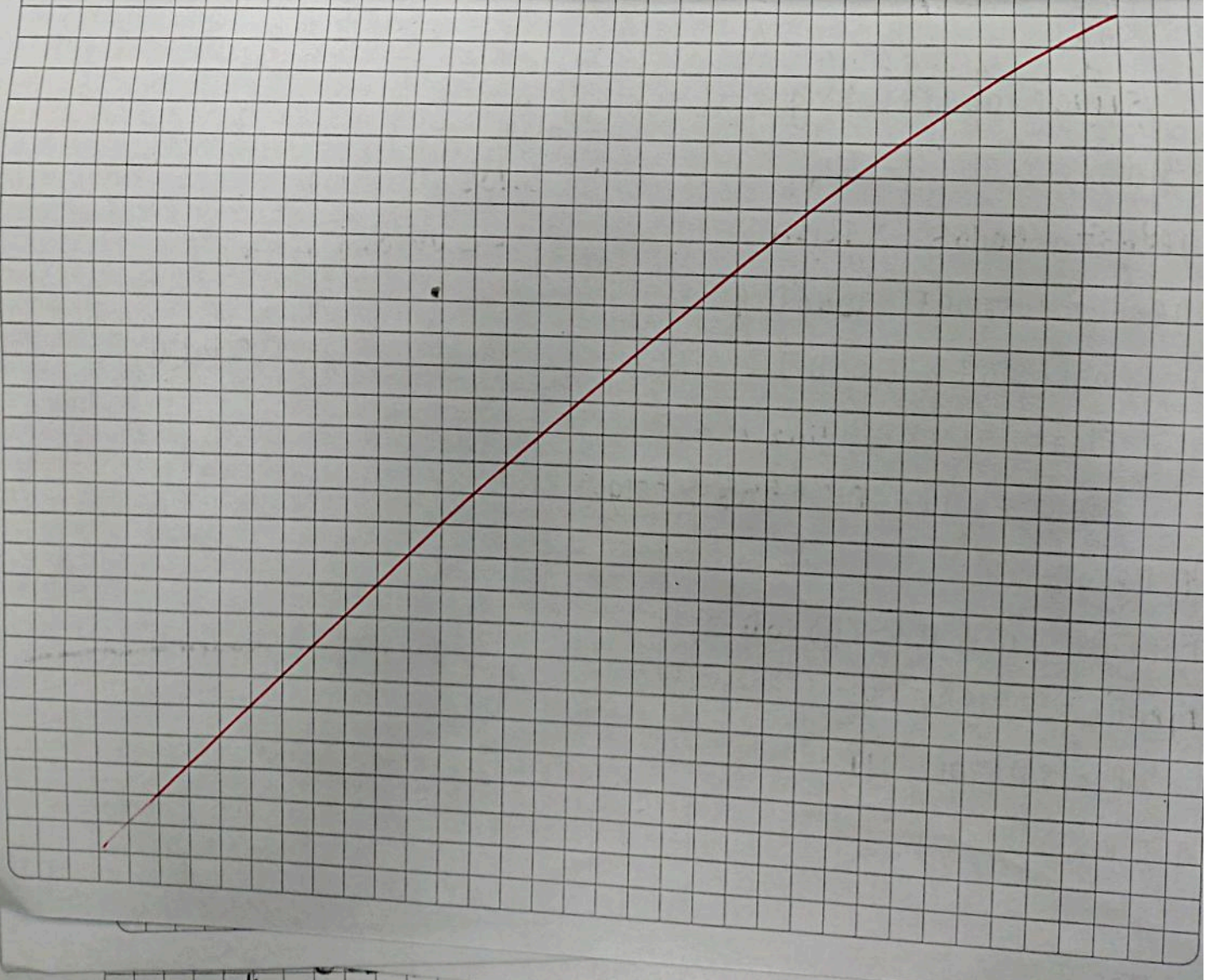
```
#include <stdio.h>   #include <stdlib.h>   #include <string.h>
#include "colas.h"
#define COD 16       #define FEC 11
```

No puede poner todo en la misma línea

```
typedef struct nodito {
    char fecha[FEC];
    float precioU, cantComp;
    struct nodito *sig;
} nodito;
typedef nodito *sublista;
typedef struct nodoS {
    char codB[COD];
    float precioUV, stock, ganan;
    sublista sub;
    struct nodoS *sig;
} nodoS;
typedef nodoS *TlistaS;
typedef struct nodoD {
    char codB[COD];
    float stock;
    struct nodoD *ant, *sig;
} nodoD;
typedef nodoD *PnodoD;
typedef struct {
    PnodoD pri, ult;
} TlistaD;
```

```
int main () {
    TLISTAD LD;        char nombArch [] = "PROVEEN.TXT";
    ULISTAS LS;
    TCola C;

    LD.pri = LD.ULT = LS = NULL;
    iniciaC (&C);

    cargaProd (&LS);
    cargaNeg (&LD);
    cargaComp (&C);

    ejerA (LS, &C, &LD);

    ejerB (nombArch, &LS, &LD);

    return 0;
}
```

} Puedes hacerlo donde las cargas!!



iii)

COLAS.H

```
#define COD 16        #define MAX 50
typedef STRUCT {
    char codB[COD];
    FLOAT CANTC;
} TElemC;
typedef STRUCT {
  TElemC datos[MAX];
  INT PRI, ULT;
} TCola;
```

COLAS.C

```
#include "colas.h"
INT VaciaC ( TCola C ) {
    return C.pri == -1;
}
void SacaC (TCola *C, TElem *X) {
  IF (!VaciaC(*C)) {
      *X = (*C).datos [ (*C).pri ];
      IF ((*C).pri == (*C).ULT)
          (*C).pri = (*C).ULT = -1;
      else
          (*C).pri++;
  }
}
```