Primer Parcial - Programación 2 - Fecha: 19/02/2004

1)

Escriba un programa que permita procesar la información de una **lista** dinámica (la información se supone ya cargada) .

En la **lista** se almacenó información con el nombre del equipo (hasta 30 caracteres válidos), el puntaje obtenido (entero), y la marca de procesado como **no-procesado** (n) .

Mediante el uso de un menú, si la opción ingresada es :

<'**1**'> : ingresar nueva información en la **lista** (con la marca de procesado en 'n'), al final de la misma .

<'**2**'> : almacenar la información del primer nodo de la **lista** en un archivo binario (<**elegidos**>), eliminándolo de la misma .

<'**3**'> : saltearse el primero de la **lista** eliminándolo del comienzo de la misma y agregándolo al final (marca de procesado : ya procesado) .

<'4'> : terminar, con lo cual se generará un archivo binario (<**sobrantes**>) con la información de los que aún quedan en la **lista** (eliminándolos de la misma) .

Para que el operador pueda decidir entre las 2 primeras opciones, en todo momento se mostrará (si la hay) la información del primero de la **lista** y un mensaje de que ya fue o no procesado, antes de mostrar las opciones del menú .

Los archivos deben ser abiertos antes de procesar la informacion .

Funciones a desarrollar (con los argumentos adecuados) :

```
ver_primero(…);          /* recupera la informacion del primero de la lista sin
                         eliminarlo de la misma */
mostrar(…);              /* muestra la información recuperada por la función
                         anterior */
menu(…);                 /* muestra las opciones de menu e ingresa (y devuelve)
                         una opcion válida */
ingresa(…);              /* permite el ingreso de la información (si es que se la
                         ingresa) a insertar en la lista */
enlistar_atras(…);       /* inserta al final de la lista la información que
                         recibe por argumento */
desenlistar_primero(…);  /* recupera la información del primer nodo, eliminándolo
                         de la lista */
grabar(…);               /* graba la información recuperada por la función
                         anterior en un archivo, recibiendo ambos por argumento
                         ATENCION :  NO almacena la marca de procesado */
```

Función Opcional : antes de comenzar el ciclo del programa, sería apreciable disponer de la lista ordenada en forma descendente por puntaje, y podrá estar en una opción extra del menú, o ejecutarse por única vez al comenzar el programa .

```
ordenar_lista(…);        /* ordena la lista en forma descedndente por orden de
                         puntaje (hace intercambio de direcciones de memoria, no
                         de información) */
```

2)

a.- Escriba una versión de la función :

```
int strncmpi(const char *s, const char *t, size_t n);
```

que compara los primeros **n** carácteres de dos cadenas que recibe por argumento, empleando lógica y aritmética de punteros . No emplear funciones de biblioteca (salvo que las desarrolle) .

b.- Función Opcional :Escriba una función recursiva `void al_reves(char *s);` que recibiendo una cadena de carácteres, la muestre al revés .

No emplear ni suponer variables globales .
Las funciones deben recibir punteros (consulte) .

Primera parte

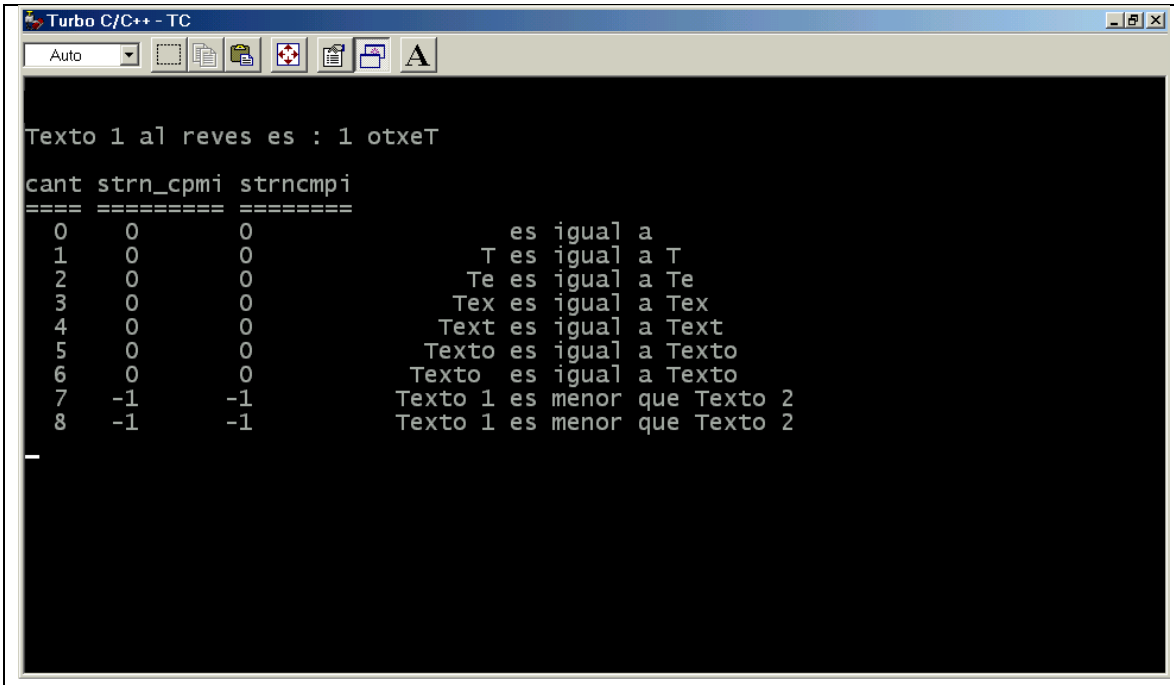
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

/*
 * Escriba un programa que permita procesar la información de una lista
 *    dinámica (la información se supone ya cargada) .
 * En la lista se almacenó información con el nombre del equipo (hasta 30
 *    carácteres válidos), el puntaje obtenido (entero), y la marca de
 *    procesado como no-procesado (n) .
 */
typedef struct
{
   struct
   {
      char        equipo[31];    /* nombre del equipo */
      int         puntaje;       /* puntaje obtenido */
   }              reg;
   char           marca;         /* marca de procesado */
} t_info;

typedef struct s_nodo
{
   t_info         info;
   struct s_nodo *sig;
} t_nodo;

/*
 * Mediante el uso de un menú, si la opción ingresada es :
 *    <'1'>  :  ingresar nueva información en la lista (con la marca de
 *       procesado en 'n'), al final de la misma .
 *    <'2'>  :  almacenar la información del primer nodo de la lista en un
 *       archivo binario (<elegidos>), eliminándolo de la misma .
 *    <'3'>  :  saltearse el primero de la lista eliminándolo del comienzo
 *       de la misma y agregándolo al final (marca de procesado :
 *       ya procesado) .
 *    <'4'>  :  terminar, con lo cual se generará un archivo binario
 *       (<sobrantes>) con la información de los que aún quedan en la lista
 *       (eliminándolos de la misma) .
 * Para que el operador  pueda decidir entre las 2 primeras opciones, en
 *    todo momento se mostrará (si la hay) la información del primero de la
 *    lista y un mensaje de que ya fue o no procesado, antes de mostrar las
 *    opciones del menú .
 * Los archivos deben ser abiertos antes de procesar la informacion .
 * Funciones a desarrollar (con los argumentos adecuados) :
 */

/*****************************************************************************
 * por estar muy apurado, desarrollo las funciones acá
 */
/*
 * recupera la informacion del primero de la lista sin eliminarlo de la misma
 */
void ver_primero(t_nodo *p, t_info *d)
{
   *d = p->info;
}

/*
 * muestra la información recuperada por la función anterior
 */
void mostrar(t_info *d)
{
   printf("\nPrimero de la lista\n"
          "Equipo  : %s\n"
```

```
          "Puntaje : %d\n"
          "%s procesado\n\n",
          d->reg.equipo,
          d->reg.puntaje,
          d->marca == 's' ? "Ya fue" : "Aún NO");
}

/*
 * muestra las opciones de menu e ingresa (y devuelve) una opcion válida
 */
int  menu(void)
{
   int op;
   do
   {
      printf("\n*** Opciones ***\n"
             "0 - Ordenar\n"
             "1 - Alta\n"
             "2 - Elegir\n"
             "3 - Pendiente\n"
             "4 - Terminar\n\n"
             "Opción : ");
      fflush(stdin);
      scanf("%d", &op);
   } while(op < 0 || op > 4);
   return op;
}

/*
 * permite el ingreso de la información (si es que se la ingresa) a insertar
 * en la lista
 */
int  ingresa(t_info *d)
{
   char          *aux;

   printf("\nPara no cargar no ingrese nada\n"
          "Equipo  :  ");
   fflush(stdin);
   fgets(d->reg.equipo, sizeof(d->reg.equipo), stdin);
   aux = strchr(d->reg.equipo, '\0');
   while(aux > d->reg.equipo && (iscntrl(*aux) || isspace(*aux)))
   {
      *aux = '\0';
      aux--;
   }
   if(!strlen(d->reg.equipo))
      return 0;

   printf("\nPara no cargar ingrese puntaje negativo\n"
          "Puntaje :  ");
   fflush(stdin);
   scanf("%d", &d->reg.puntaje);
   if(0 > d->reg.puntaje)
      return 0;

   return 1;
}

/*
 * inserta al final de la lista la información que recibe por argumento
 */
void enlistar_atras(t_nodo **p, t_info *d)
{
   while(*p)
      p = &(*p)->sig;
```

```
   if((*p = (t_nodo *)malloc(sizeof(t_nodo))) == NULL)
      exit(fprintf(stderr, "Memoria insuficiente"));

   (*p)->info = *d;
   (*p)->sig  = NULL;
}

/*
 * recupera la información del primer nodo, eliminándolo de la lista
 */
void desenlistar_primero(t_nodo **p, t_info *d)
{
   t_nodo        *elim = *p;
   *d = elim->info;
   *p = elim->sig;
   free(elim);
}

/*
 * graba la información recuperada por la función anterior en un archivo,
 * recibiendo ambos por argumento
 * ATENCION :  NO almacena la marca de procesado
 */
void grabar(FILE *fp, t_info *d)
{
   fwrite(&d->reg, sizeof(d->reg), 1, fp);
}


/*****************************************************************************
 * Función Opcional :  antes de comenzar el ciclo del programa, sería
 *    apreciable disponer de la lista ordenada en forma descendente por
 *    puntaje, y podrá estar en una opción extra del menú, o ejecutarse
 *    por única vez al comenzar el programa .
 */
/*
 * ordena la lista en forma descedente por orden de puntaje (hace intercambio
 * de direcciones de memoria, no de información)
 */
void ordenar_lista(t_nodo **p)
{
   t_nodo        *act,
                 **ant;
   int            marca          = 1;

   while(marca)
   {
      marca = 0;
      ant = p;
      act = *p;
      while(act && act->sig)
      {
         if(act->info.reg.puntaje < act->sig->info.reg.puntaje)
         {
            marca = 1;
            *ant = act->sig;
            act->sig = (*ant)->sig;
            (*ant)->sig = act;
            act = *ant;
         }
         ant = &act->sig;
         act = act->sig;
      }
   }
}
```

```
/*****************************************************************************
 * función :   main
 */
void main(void)
{
   t_info         info;
   t_nodo        *lista;
   int            opcion;
   FILE          *elegidos,
                 *sobrantes;
   /*
    * empiezo por abrir los archivos involucrados
    */
   if((elegidos = fopen("elegidos", "wb")) == NULL)
      exit(fprintf(stderr, "Error archivo elegidos"));
   if((sobrantes = fopen("sobrantes", "wb")) == NULL)
      exit(fprintf(stderr, "Error archivo sobrantes"));

   /*
    * acá se inicializa la lista y se carga la información
    */
   lista = NULL;  /* se supone que en algún momento la lista puede quedar
                   *    vacía, con lo cual con esto basta para probar */

   ordenar_lista(&lista);

   do
   {
      if(lista != NULL)    /* como no me lo exigen no invoco lista_vacia */
      {
         ver_primero(lista, &info);
         mostrar(&info);
      }
      else
         puts("\nLa lista está vacía\n");
      switch(opcion = menu())
      {
      case 0:
            ordenar_lista(&lista);
            break;
      case 1:
            if(ingresa(&info))
            {
               info.marca = 'n';
               enlistar_atras(&lista, &info);
            }
            break;
      case 2:
            if(lista != NULL)
            {
               desenlistar_primero(&lista, &info);
               grabar(elegidos, &info);
            }
            break;
      case 3:
            if(lista != NULL)
            {
               desenlistar_primero(&lista, &info);
               info.marca = 's';
               enlistar_atras(&lista, &info);
            }
      }
   } while(opcion != 4);

   while(lista != NULL)
   {
      desenlistar_primero(&lista, &info);
```

```
        grabar(sobrantes, &info);
    }

    fclose(elegidos);
    fclose(sobrantes);
}
```




```
/*****************************************************************************
 * a.- Escriba una versión de la función :
 *       int strncmpi(const char *s, const char *t, size_t n);
 *    que compara los primeros n carácteres de dos cadenas que recibe por
 *    argumento, empleando lógica y aritmética de punteros .
 *    No emplear funciones de biblioteca (salvo que las desarrolle) .
 * b.- Función Opcional :Escriba una función recursiva
 *       void al_reves(char *s);
 *    que recibiendo una cadena de carácteres, la muestre al revés .
 */

#include <stdio.h>
#include <string.h>

int to_upper(int c)
{
   if(c >= 'a' && c <= 'z')
      c -= 32;
   return c;
}

int strn_cmpi(const char *s, const char *t, size_t n)
{
   if(n)
   {
      n--;
      while(*s && to_upper(*s) == to_upper(*t) && n)
      {
         s++;
         t++;
         n--;
      }
      return to_upper(*s) - to_upper(*t);
   }
   return 0;
}

void al_reves(const char *s)
{
```

```
   if(*s)
   {
      al_reves(s + 1);
      putchar(*s);
   }
}

void main(void)
{
   char           Texto1[]       = {"Texto 1"},
                  Texto2[]       = {"Texto 2"};
   int            cmp,
                  ciclo;

   printf("%s al reves es : ", Texto1);
   al_reves(Texto1);

   puts("");

   printf("\n"
          "cant strn_cpmi strncmpi\n"
          "==== ========= ========\n");
   for(ciclo = 0; ciclo <= sizeof(Texto1); ciclo++)
   {
      cmp = strn_cmpi(Texto1, Texto2, ciclo);
      printf("%3d %4d %7d         ",
             ciclo, cmp, strncmpi(Texto1, Texto2, ciclo));
      if(cmp == 0)
         printf("%*.*s es igual a %.*s\n",
                sizeof(Texto1), ciclo, Texto1,
                ciclo, Texto2);
      else
         if(cmp < 0)
            printf("%*.*s es menor que %.*s\n",
                   sizeof(Texto1), ciclo, Texto1,
                   ciclo, Texto2);
         else
            printf("%*.*s es mayor que %.*s\n",
                   sizeof(Texto1), ciclo, Texto1,
                   ciclo, Texto2);
   }
}
```