Ejercicios Resueltos

Cadenas de caracteres

1.- Realiza una función int posición(char * cad, char c) que devuelva la primera posición de la cadena cad en aparece el carácter c o -1 si no aparece.

1ª versión (usando memoria estática y aritmética de punteros)

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

int posicion (char *,char);

int main()
{
   char car;
   int resp;
   char cad[10];
   puts ("Introduce cadena < 10 caracteres");
   gets(cad);
   do{
      fflush(stdin);
      puts ("Introduce caracter a buscar en cad");
      car=getchar();
      resp=posicion (cad,car);
      if (resp== -1)
          printf ("El caracter %c no existe en \"%s\"\n",car,cad);
      else
          printf("El caracter %c está en la posición %d de:\"%s\"\n",car,resp
          +1,cad);
      fflush(stdin);
      puts("¿Otro caracter a buscar? (s/n)");
      scanf("%c",&resp);
   } while (resp=='s' || resp=='S');     //Repetir la búsqueda
   system("pause");
   return 0;
}
```

```
//cabecera:   int posicion ( char *cade, char c)
//precondición:  cade y c inicializados, c es caracter a buscar
//postcondición:  devuelve la posición de la primera aparición del caracter c en la cadena cade o -1 si no existe.

int posicion (char *cade,char c)//puede usarse cade[ ] indistintamente
{
int i=0,j;
while (*(cade+i)!='\0' && *(cade +i) !=c)
            i++;
if (*(cade +i)==c)
   j=i;
else
   j=-1;
return j;
}
```

2ª versión (usando memoria dinámica y notación [ ])

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

int posicion(char [],char);

int main()
{
 char car;
 int n,resp;
char *cad;
do{
    puts ("Introduce tamaño de la cadena");
    scanf("%d",&n);
   }while (n<1);
if ((cad =(char *)malloc(n*sizeof(char)))== NULL)
       puts ("No se ha realizado la reserva de memoria");   //Reserva dinámica
else
{
   fflush(stdin);
   puts("Introduzca cadena");
   gets(cad);
   do{
       fflush(stdin);
       puts ("Introduce caracter a buscar en cad");
       car=getchar();
       resp= posicion(cad,car);
       if (resp==-1)
```

```
        printf ("El caracter %c no existe en \"%s\"\n",car,cad);
      else
        printf("El caracter %c está en la posición %d de:\"%s\"\n",car,resp +1,cad);
        fflush(stdin);
        puts("¿Otro caracter a buscar? (s/n)");
        scanf("%c",&resp);
    }while (resp=='s' || resp=='S');  //Repetir la búsqueda
 }
system("pause");
free(cad);
return 0;
}


//cabecera int posicion( char cade[], char c)
//precondición cade y c inicializados- c es caracter a buscar
//postcondición devuelve la posición de la primera aparición de c en cade o -1 si no existe
int posicion(char cade[],char c)//puede usarse *cade indistintamente
{
 int i=0,j;
 while (cade[i]!='\0' && cade[i] !=c)
            i++;
 if (cade[i]==c)
    j=i;
 else
    j=-1;
 return j;
}
```

Matrices

2.- (ejercicio 15 de la relación de problemas 4 de prácticas)Un cuadrado latino de orden N es una matriz cuadrada que en su primera fila contiene los N primeros números naturales y en cada una de las siguientes N-1 filas contiene la rotación de la fila anterior un lugar a la derecha. Realiza un programa que reciba como parámetro la dimensión N y genere la matriz correspondiente a su cuadrado latino.

```
//Resuelve el problema reservando memoria dinámicamente para la matriz
#include <stdio.h>
#include <stdlib.h>

void cuadradolatino(int , int **);

int main()
{
  int n,i,j;
```

```
 int **m;
 printf("Introduce el tamaño del cuadrado latino: ");
 scanf("%i",&n);
 if ((m=(int**)malloc(sizeof(int*)*n))== NULL)
 {
      printf("Error al asignar memoria");
      system("pause");
      exit(1);
 }
 for(i=0;i<n;i++)
     if((m[i]=(int*)malloc(sizeof(int)*n))==NULL)
     {
        printf("Error al asignar memoria");
        system("pause");
        exit(1);
     }
 cuadradolatino(n,m);
 for(i=0;i<n;i++)// Se escribe por pantalla la matriz generada
{
    for(j=0;j<n;j++)
        printf("%i ",m[i][j]);
    printf("\n");
 }
 for (i=0;i<n;i++)//Liberamos la memoria reservada para la matriz
   free(m[i]);
 free(m);
 system("pause");
 return(0);
}

//cabecera: void cuadradolatino(int n, int **mat)
//precondicion: recibe la matriz vacía y su dimensión
//poscondición: genera el cuadrado latino en la matriz que recibe
void cuadradolatino(int n, int **mat)
{
 int f,c;

 for(f=0;f<n;f++)
   for(c=f;c<n+f;c++)
      if(c>n-1)
         *(*(mat+f)+c-n)=c-f+1; //o bien mat[f][c-n]=c-f+1;
      else
         *(*(mat+f)+c)=c-f+1;//o bien mat[f][c]=c-f+1;
}
```

3.- (ejercicio 8 de la relación 5 de teoría) Se dice que una matriz de dos dimensiones posee un punto de silla en una cierta posición, si el valor de dicha posición es mayor que el de todas las de su fila, y menor que el de todas las de su columna, o viceversa (si el valor de dicha posición es menor que el de todas las de su fila, y mayor que el de todas las de su columna). Diseña un algoritmo en pseudocódigo que lea una matriz de NxM elementos enteros (N y M conocidos), y muestre por pantalla cuántos puntos de silla contiene la matriz y las posiciones que ocupan.

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

// Se definen las macros con el tamaño de la matriz
#define N 3
#define M 3

//Prototipos de las funciones
int pmayor(int [N][M],int, int);
int pmenor(int [N][M],int, int);
void puntossilla(int [N][M]);
void leermatriz(int [N][M]);

int main()
{
  //Declaración de la matriz con tamaño conocido
   int matriz[N][M];
   leermatriz(matriz);
   puntossilla(matriz);
   puts("\n");
   system("pause");
   return(0);
}

//cabecera: void leermatriz(int matriz[N][M])
//precondición: recibe la matriz sin inicializar
//poscondición: devuelve la matriz rellena
void leermatriz(int matriz[N][M])
{
   int fila,columna;
   for(fila=0;fila<N;fila++)
   {
      for(columna=0;columna<M;columna++)
        {
         printf("\nIntroduce el valor de la posición: Fila %i Columna %i
         :",fila,columna);
         scanf("%i",&matriz[fila][columna]);
         }
 }
```

```
}

//cabecera: int pmayor(int matriz[N][M],int f,int c)
//precondición: recibe la matriz inicializada y la fila y la columna del elemento
//poscondición: devuelve 1 si el numero en la matriz en la posición f,c es mayor que cualquiera de los de su fila y menor que cualquiera de su columna

int pmayor(int matriz[N][M],int f,int c)
{
//Suponemos que es un punto de silla
int psilla=1;
int i;
for(i=0;(i<N && psilla==1);i++)
 {
  //si encuentra uno mayor en la misma fila o uno menor en la misma columna,
  //entonces no es punto de silla
   if( (matriz[f][i]>matriz[f][c]) || (matriz[i][c]<matriz[f][c])  )
          psilla=0;
  }
return(psilla);
}

//cabecera: int pmayor(int matriz[N][M],int f,int c)
//precondición: recibe la matriz inicializada y la fila y la columna del elemento
//poscondición: devuelve 1 si el numero en la matriz en la posición f,c es menor que cualquiera de los de su fila y mayor que cualquiera de su columna
int pmenor(int matriz[N][M],int f,int c)
{
  //Suponemos que es un punto de silla
  int psilla=1;
  int i;
  for(i=0;(i<N&&psilla==1);i++)
  {
   //...si encuentra uno menor en la misma fila o uno mayor en la misma columna,
   //entonces no es punto de silla
  if( (matriz[f][i]<matriz[f][c]) || (matriz[i][c]>matriz[f][c])  )
        psilla=0;
   }
 return(psilla);
}
```

```
//cabecera: void puntosilla(int matriz[N][M])
//precondición: recibe la matriz inicializada
//poscondición: comprueba si cada posición de una matriz NxM es un punto de silla
void puntossilla(int matriz[N][M])
{
 int fila,columna;
 for(fila=0;fila<N;fila++)
 {
    for(columna=0;columna<M;columna++)
     {
        //Comprobamos si el número en la posición fila,columna es un punto de silla
        if(pmayor(matriz,fila,columna) || pmenor(matriz,fila,columna) )
                printf("\nSe ha encontrado un punto de silla en la posición: Fila  %i
                Columna %i :",fila,columna);
        }
  }
}
```

Estructuras (registros)

4.- (30 de la relación de problemas 4 de prácticas) Un instituto desea almacenar los datos de los N alumnos de un curso. De cada alumno se almacenaran nombre, apellidos, dni, edad y las calificaciones que ha obtenido en cada uno de los tres trimestres de cada una de las 7 asignaturas del curso; junto con las calificaciones deberá almacenarse también el nombre de la asignatura. Diseña la estructura necesaria para almacenar dicha información e implementa las siguientes funciones:

- void nota_global (alumno * al) Recibe como parámetro exclusivamente los datos de un alumno y debe escribir por pantalla la nota global para cada asignatura.

- void mayor_nota (char *asig, alumno *clase) Recibe como parámetro los datos de todos los alumnos y el nombre de una asignatura y deberá escribir por pantalla el nombre y apellidos del alumno que mas nota haya obtenido en la asignatura especificada por asig durante el primer trimestre.

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

#define N 3

typedef struct{
            char nombre[15];
            float nota;
            }asignatura;
```

```
typedef struct{
		char nombre[10];
		char apellidos[30];
		char dni[10];
		int edad;
		asignatura notas_asignaturas[3][3];
		} alumno;

void inicializa(alumno alumnos[]);
void nota_global(alumno *a);
void mayor_nota (char * asig, alumno * curso);

int main()

{
 alumno alumnos_curso[N];
 char dni[10];
 int i,encontrado=0;
 char asignatura[15];
 inicializa(alumnos_curso);
 printf("escriba dni del alumno del que desea conocer la nota\n");
 gets(dni);
 fflush(stdin);
 for (i=0;(i<N&&encontrado ==0);i++)
    if(strcmp(dni, alumnos_curso[i].dni)==0)
       encontrado=1;
 if (encontrado ==1)
   nota_global(&alumnos_curso[i]);
 else
   printf("El alumno buscado no existe\n");
 printf("Escriba el nombre de una asignatura\n");
 gets(asignatura);
 fflush(stdin);
 mayor_nota (asignatura, alumnos_curso);
 system("pause");
 return 0;
}

//cabecera: void inicializa(alumno alumnos[])
//precondicion: recibe el vector de estructuras sin inicializar
//poscondicion: inicializa el vector de estructuras con los valores insertados por el usuario
void inicializa(alumno alumnos[])
{
 int i,j,k;
 for (i=0;i<N;i++)
 {
  printf("escriba nombre del alumno\n");
```

```
 gets(alumnos[i].nombre);
 fflush(stdin);
 printf("escriba apellidos\n");
 gets(alumnos[i].apellidos);
 fflush(stdin);
 printf("escriba dni\n");
 gets(alumnos[i].dni);
 fflush(stdin);
 printf("escriba edad\n");
 scanf("%d", &(alumnos[i].edad));
 fflush(stdin);
 for (j=0;j<3;j++)
   for (k=0;k<3;k++)
    {
     printf("escriba nombre asignatura, trimestre %d", k);
     gets(alumnos[i].notas_asignaturas[j][k].nombre);
     fflush(stdin);
     printf("escriba nota asignatura, trimestre %d", k);
     scanf("%f",&(alumnos[i].notas_asignaturas[j][k].nota));
     fflush(stdin);
    }

  }
}
//cabecera: void nota_global(alumno *a)
//precondición: recibe la estructura de un alumno
//poscondición: Escribe la nota global del alumno en cada asignatura
void nota_global(alumno *a)
{
 float nota=0;
 int j,k;
 for (j=0;j<3;j++)
  {
    for (k=0;k<3;k++)
     nota=nota + a->notas_asignaturas[j][k].nota;
    printf("La nota en la asignatura %s es %f\n", a->notas_asignaturas[j][k-
1].nombre, nota);
    nota=0;
  }
}
//cabecera: void mayor_nota (char * asig, alumno * curso)
//precondicion: recibe el nombre de una asignatura y el vector de estructuras completo
//poscondición: escribe nombre y apellidos del alumno con mayor nota en esa asignatura durante el
primer trimetre
void mayor_nota (char * asig, alumno * curso)
{
 int i, j,pos, enc=0;
```

```
 float max=0.0;
 for (i=1;(i<N&&enc==0); i++)
   for (j=1;(j<3&&enc==0);j++)
     if (strcmp(asig,curso[i].notas_asignaturas[j][0].nombre)==0)
        {
         max=curso[i].notas_asignaturas[j][0].nota;
         enc=1;
         }

 for (i=0;i<N; i++)
   for (j=0;j<3;j++)
     if                    ((strcmp(asig,curso[i].notas_asignaturas[j][0].nombre)==0)&&
(curso[i].notas_asignaturas[j][0].nota>max))
        {
        max=curso[i].notas_asignaturas[j][0].nota;
        pos=i;
      }
  printf("El alumno con mas nota es %s, %s\n", curso[pos].nombre,
curso[pos].apellidos);
}
```

Ficheros

5.- (ejercicio 33 de la relación 4 de prácticas) Realiza un programa que permita a través de un menú realizar las siguientes operaciones sobre un fichero de texto: crear un fichero, escribir en el fichero sin eliminar su contenido, leer el contenido y escribirlo en pantalla.

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

int main()
{

 int opcion=-1;
 char c;
 FILE *fichero;

 do{
        //Muestra el menú
        printf("Selecciona opción\n\n");
        printf("1. Crear un fichero\n");
        printf("2. Escribir en el fichero sin eliminar el contenido\n");
        printf("3. Leer el contenido y escribir en pantalla\n\n");
        printf("0. Salir\n");
```

```
//Pide una opción
scanf("%i",&opcion);
//En función de la opción introducida....
switch(opcion){
   //Sale del programa
   case 0:{
            exit(0);
           }
  //Crea un fichero llamado pratica.txt
   case 1:{
             if ((fichero=fopen("practica.txt","wt"))==NULL)
            {
              printf("Error al abrir fichero\n");
              exit(1);
            }
            else
            {
              printf("fichero creado\n");
              system("pause");
                 fclose(fichero);
            }
              break;
           }
  //Escribe contenido en el fichero sin eliminar lo ya existente en el
   case 2:{
             if( (fichero=fopen("practica.txt","at+"))==NULL)
           {
                printf("Debe crear el fichero previamente\n");
            }
            //Escribe una cadena de caracteres en el fichero
            printf("Introduzca caracteres en el fichero y '*' para terminar\n");
            while ((c=getchar())!= '*')
                 fputc(c, fichero);
          fclose(fichero);
            break;
          }
 //Lee el contenido del fichero y escribe el contenido en pantalla
   case 3:{
              if( (fichero=fopen("practica.txt","rt"))==NULL)
             {
                printf("Debe crear el fichero previamente\n");
              }
               //Mientras el caracter leido no sea EOF, lee un caracter y lo
               imprime en pantalla
               printf("El contenido del fichero es:\n");
             while((c=fgetc(fichero))!=EOF)
```

```
                    printf("%c",c);
                  printf("\n");
                  fclose(fichero);
                  system("pause");
                  break;
                 }
               }
          system("cls");

     }while(opcion!=0);
}
```

Memoria dinámica

6.- (ejercicio 37 de la relación de problemas 4 de prácticas) Dado un vector dinámico de n elementos inicializados con valores 0 y 1. Escribe una función que reciba dicho vector y el tamaño n y añada un elemento más que represente el bit de paridad, es decir que contenga 0 o 1 para que la paridad sea par, es decir que la cantidad total de 1 sea par.

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

void asigna_paridad(int **vector, int n);

#define TAM   10

int main(void)
{
 int *vector, i;

 vector = (int *) malloc(TAM * sizeof(int));
 if (vector == NULL) {
   puts("No hay suficiente memoria dinámica");
   exit(1);
 }
 for (i=0; i<TAM; i++)
   vector[i] = i % 2;
 asigna_paridad(&vector, TAM);
 free(vector);
 system("pause");
 return 0;
}
```

*// cabecera: void asigna_paridad(int **vector, int n)*
//precondición: recibe el vector dinámico inicializado con valores 0 y 1
//poscondicion: asigna la paridad aumentando para ello de forma dinámica una posición en el vector para el bit de paridad

```
void asigna_paridad(int **vector, int n)
{
 int *nuevo, i, unos;

 nuevo = (int *) realloc(*vector, (n+1)*sizeof(int));
 if (nuevo == NULL) {
   puts("No hay suficiente memoria dinámica");
   exit(1);
 }
 for (unos=i=0; i<n; i++)
   if (nuevo[i])
     ++unos;
 nuevo[i] = unos % 2;
 *vector = nuevo;
}
```

7.- (ejercicio 38 de la relación de problemas 4 de prácticas) Escribe un programa que cree una matriz de forma dinámica pidiendo al usuario la dimensión de las filas y columnas de la matriz. Posteriormente debe introducir valores en la matriz e imprimir su resultado.

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

int main(void)
{
 int filas, columnas, i, j;
 int **matriz;

 /* Lectura del tamaño de la matriz */
 printf("Introduzca las filas: ");
 scanf("%d", &filas);
 fflush(stdin);
 printf("Introduzca las columnas: ");
 scanf("%d", &columnas);
 fflush(stdin);

 /* Reserva de memoria para la matriz */
 matriz = (int **) malloc(filas*sizeof(int *));
 if (matriz == NULL)
{
   puts("No hay suficiente memoria dinámica");
```

```
   exit(1);
  }
  for (i=0; i<filas; i++)
{
   matriz[i] = (int *) malloc(columnas*sizeof(int));
   if (matriz[i] == NULL)
   {
     puts("No hay suficiente memoria dinámica");
     exit(1);
   }
  }

  /* Introducción de datos en la matriz */
  for (i=0; i<filas; i++)
    for (j=0; j<columnas; j++)
      matriz[i][j] = i*100+j;

  /* Visualización de los datos de la matriz */
  for (i=0; i<filas; i++)
 {
    for (j=0; j<columnas; j++)
      printf("%04d  ", matriz[i][j]);
    putchar('\n');
  }
   /*Liberación de la memoria dinámica*/
  for (i=0;i<filas ;i++)
      free(matriz[i]);
  free(matriz);
  system("pause");
  return 0;
}
```

8.- (ejercicio 39 de la relación de problemas 4 de prácticas) Realiza el ejercicio anterior usando una función que reserve memoria para la matriz en lugar de hacerlo en la función main.

- **Versión 1**

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

int **malloc_matriz(int f, int c);
void free_matriz(int **m, int f);

int main(void)
{
  int filas, columnas, i, j;
```

```
  int **matriz;

  /* Lectura del tamaño de la matriz */
  printf("Introduzca las filas: ");
  scanf("%d", &filas);
  fflush(stdin);
  printf("Introduzca las columnas: ");
  scanf("%d", &columnas);
  fflush(stdin);

  matriz = malloc_matriz(filas, columnas);

  /* Introducción de datos en la matriz */
  for (i=0; i<filas; i++)
    for (j=0; j<columnas; j++)
      matriz[i][j] = i*100+j;

  /* Visualización de los datos de la matriz */
  for (i=0; i<filas; i++) {
    for (j=0; j<columnas; j++)
      printf("%04d  ", matriz[i][j]);
    putchar('\n');
  }

  free_matriz(matriz, filas);

  system("pause");
  return 0;
}
//cabecera: int ** malloc_matriz (int f, int c)
//precondicion: recibe el numero de filas y columnas de la matriz y devuelve la matriz reservada dinámicamente
//poscondicion: devuelve la matriz reservada dinámicamente
int **malloc_matriz(int f, int c)
{
  int **m, i;

  m = (int **) malloc(f*sizeof(int *));
  if (m == NULL) {
    puts("No hay suficiente memoria dinámica");
    exit(1);
  }
  for (i=0; i<f; i++) {
    m[i] = (int *) malloc(c*sizeof(int));
    if (m[i] == NULL) {
      puts("No hay suficiente memoria dinámica");
      exit(1);
```

```
    }
  }
  return m;
}

//cabecera: void free_matriz(int **m, int f)
//precondición: recibe la matriz y el numero de filas
//poscondicion: libera la memoria reservada para la matriz
void free_matriz(int **m, int f)
{
  int i;

  for (i=0; i<f; i++)
    free(m[i]);
  free(m);
}
```

- **Versión 2**

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

void reserva_matriz(int ***m, int f, int c);
void free_matriz(int **m, int f);

int main(void)
{
  int filas, columnas, i, j;
  int **matriz;

  /* Lectura del tamaño de la matriz */
  printf("Introduzca las filas: ");
  scanf("%d", &filas);
  fflush(stdin);
  printf("Introduzca las columnas: ");
  scanf("%d", &columnas);
  fflush(stdin);

  reserva_matriz(&matriz, filas, columnas);

  /* Introducción de datos en la matriz */
  for (i=0; i<filas; i++)
    for (j=0; j<columnas; j++)
      matriz[i][j] = i*100+j;
```

```
 /* Visualización de los datos de la matriz */
  for (i=0; i<filas; i++) {
    for (j=0; j<columnas; j++)
      printf("%04d  ", matriz[i][j]);
    putchar('\n');
  }

  free_matriz(matriz, filas);

  system("pause");
  return 0;
}
//cabecera: void reserva_matriz (int ***m, int f, int c)
//precondicion: recibe por referencia la matriz para hacer la reserva, el numero de filas y columnas
de la misma
//poscondicion: reserva memoria dinámicamente para la matriz
void reserva_matriz(int ***m, int f, int c)
{
  int i;

  *m = (int **) malloc(f*sizeof(int *));
  if (m == NULL) {
    puts("No hay suficiente memoria dinámica");
    exit(1);
  }
  for (i=0; i<f; i++) {
    (*m)[i] = (int *) malloc(c*sizeof(int));
    if ((*m)[i] == NULL) {
      puts("No hay suficiente memoria dinámica");
      exit(1);
    }
  }
}

//cabecera: void free_matriz(int **m, int f)
//precondición: recibe la matriz y el numero de filas
//poscondicion: libera la memoria reservada para la matriz
void free_matriz(int **m, int f)
{
  int i;

  for (i=0; i<f; i++)
    free(m[i]);
  free(m);
}
```