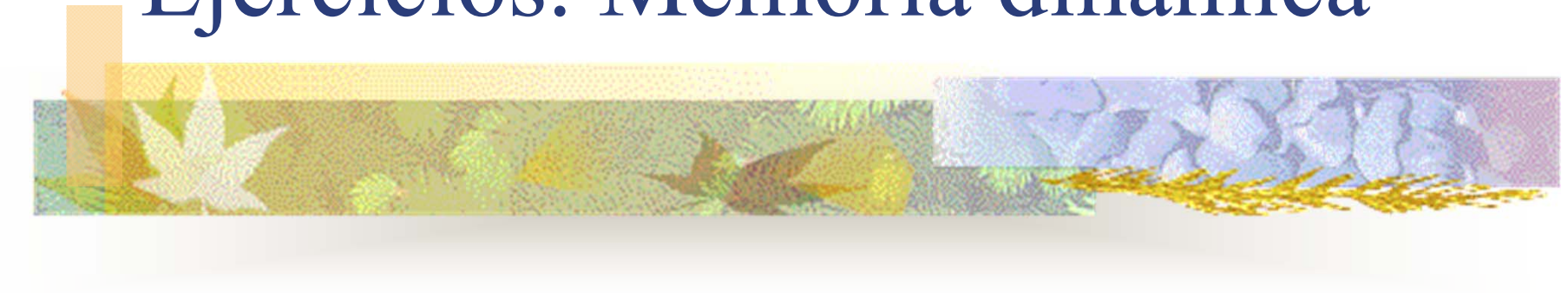
Ejercicios. Memoria dinámica



Eva Lucrecia Gibaja Galindo
Dpto. Informática y Análisis Numérico

Ejercicio 1

- ¿Qué errores hay en las siguientes declaraciones y sentencias?

```
int n, *p;
char ** dob= “Cadena dos punteros”;
p=n*malloc(sizeof(int));
```

Ejercicio 1 (Solución)

- `char ** dob= "Cadena dos punteros";`
  - Una cadena de caracteres se almacena en C como un *array* de caracteres de tipo *char*, por lo que puede referenciarse por medio de un puntero *char*, pero no como uno doble.
- `p=n*malloc(sizeof(int));`
  - No tiene sentido multiplicar la dirección que devuelve *malloc* por *n*.
  - No se está haciendo el *casting* necesario.

Ejercicio 2

- Dada la siguiente declaración de estructura, reservar memoria dinámicamente para una estructura asignando su dirección a *b*.

```
struct boton {
char * rotulo;
int codigo;
};
struct boton * b;
```

Ejercicio 2 (Solución)

- Dada la siguiente declaración de estructura, reservar memoria dinámicamente para una estructura asignando su dirección a b.

```
struct boton {
char * rotulo;
int codigo;
}
struct boton * b;
b=(struct boton *)malloc(sizeof(struct boton));
b->rotulo = (char*) malloc(100*sizeof(char));
```

Ejercicio 3

- Sentencias para leer y escribir los campos de la variable *b*.

Ejercicio 3 (Solución)

- Sentencias para leer y escribir los campos de la variable *b*.

```
printf("Rotulo: ");
scanf("%s",b->rotulo);
printf("Codigo: ");
scanf("%d",&(b->codigo));
printf("Rotulo: %s\n",b->rotulo);
printf("Codigo: %d\n",b->codigo);
```

Ejercicio 4

- ¿Qué diferencias existen entre las siguientes declaraciones?

```
char *c [15];
char ** c;
char c[15][12];
```

Ejercicio 4 (Solución)

- `char *c [15];`
  - array de 15 punteros de tipo char o cadenas de caracteres. Estos punteros no están inicializados; no apuntan a ninguna dirección de memoria válida
- `char ** c;`
  - Puntero doble a objetos de tipo char*
- `char c[15][12]`
  - Matriz de caracteres con 15 filas y 12 columnas. Tiene reservado espacio para 15*12 caracteres.`[12];`

Ejercicio 5

- Escribe las sentencias de código fuente para leer *N* líneas de caracteres y asignar cada línea a un elemento del *array* dadas las siguientes declaraciones.

```
#define N 4
char * linea[N];
```

Ejercicio 5 (Solución)

```
int i;
char temp[80];
for (i=0;i<N;i++)
{
  printf("Linea: ");
  gets(temp);
  linea[i]=(char*)malloc(sizeof(char)*(strlen(temp)+1));
  strcpy(linea[i],temp);
}
```

Ejercicio 6

- Escribe una función que reciba *N* líneas de caracteres y libere las líneas de longitud menor de 20 caracteres. Las líneas restantes han de quedar en orden consecutivo, desde la posición cero.

Ejercicio 6 (Solución)

```
void elimina(char * lineas[], int N) {
   int i,j;
   for (i=0;i<N;i++){
      if(strlen(lineas[i])<20) {
         free(lineas[i]);
         for (j=i;j<N-1;j++){
            linea[j]=linea[j+1];
         }
         linea[j+1]=NULL;
      }
   }
}
```

Ejercicio 7

- Descubre los errores

```
char *pta, car1;
*pta = car1;

int *ptc, b;
b = malloc ( sizeof ( int ) );
*b = 8;
ptc = malloc(sizeof(int));
ptc = b+5;
free(*ptc);
b = *ptc + 10;
```

Ejercicio 7 (Solución)

- Descubre los errores

```
char *pta, car1;
*pta = car1;

int *ptc, b;
b = (int*) malloc ( sizeof ( int ) );
*b = 8;
ptc = (int *) malloc(sizeof(int));
ptc = b+5;
free(*ptc);
b = *ptc + 10;
```

Ejercicio 7 (Solución)

- *pta=car1
  - uso del puntero pta sin inicializar
- b=malloc(sizeof(int));
  - b es de tipo entero y malloc devuelve un puntero
- *b=8
  - El operador * no se puede aplicar sobre algo que no sea un puntero
- ptc =b+5
  - Asignación a un puntero de una expresión de tipo entero
- free(*ptc)
  - Hay que liberar el puntero ptr no el contenido del objeto referenciado

Ejercicio 8

- Determinar el resultado que almacenan las variables al final

```
int *ptc, *ptn,  b=20;
ptc = &b;
ptn = malloc ( sizeof (int));
*ptn = b + 15;
*ptc = *ptc -1;
ptc = ptn;
*ptc = 2;
```

Ejercicio 8 (Solución)

- Determinar el resultado que almacenan las variables al final: *ptn=2 , b=19, ptn==ptc

```
int *ptc, *ptn,  b=20;
ptc = &b;
ptn = malloc ( sizeof (int));
*ptn = b + 15;
*ptc = *ptc -1;
ptc = ptn;
*ptc = 2;
```

Ejercicio 9

- Cambiar el siguiente código para utilizar aritmética de punteros

```
int AE[4] = {4,1,5,2}, *p;
int i;
p = AE;
for(i = 0; i < 4; i++){
      p[i] = p[i] + 3;
}
p = malloc(sizeof(int)*9);
for(i = 0; i < 9; i++){
      p[i] = 5;
}
```

Ejercicio 9 (Solución)

- Cambiar el siguiente código para utilizar aritmética de punteros

```
int AE[4] = {4,1,5,2}, *p;
int i;
p = AE;
for(i = 0; i < 4; i++){
        *(p+i) = *(p+i) + 3;
}
p = malloc(sizeof(int)*9);
for(i = 0; i < 9; i++){
        *(p+i) = 5;
}
```

Ejercicio 10

- Escriba un programa que realice las siguientes operaciones:
  - Declare un puntero a char llamado **B** y otro llamado **C**
  - Asígnele memoria a **B** para 30 caracteres
  - Copie en **B** una palabra cualquiera
  - Usando aritmética de punteros, modifique todos los caracteres de la cadena para que pase de minúsculas a mayúsculas
  - Usando aritmética de punteros **C** debe apuntar al carácter de la mitad de la cadena **B.** Imprima C, que sale en pantalla?

Ejercicio 10 (Solución)

```
#include <stdio.h>
#include <stdlib.h>
void main()
{    char* b, *c;
     int i;

     b= (char* )malloc(30*sizeof(char));
     strcpy(b, "hola");

     for(i=0; b[i]!='\0';i++)
        *(b+i)-='z'-'Z';
     c=b+i/2;
     printf("\nLa cadena resultado es: <%s> y el
caracter en la posicion %d es %c", b,i/2, *c);
}
```

Ejercicios 7 y 8

- Escribe una función que reciba un vector y devuelva los elementos pares del vector en otro vector.
  - **SeleccionaPares.c**
- Se tiene una matriz de 20x20 elementos enteros. Se quiere eliminar de la matriz los números 13 que haya. Escribir una función que genere, a partir de la matriz inicial, un vector dinámico de 20 vectores de tamaño arbitrario en los que no aparezca el nº 13.
  - **eliminaTrece.c**