# Manejo dinámico de la memoria

**¿Cuál es la salida de los siguientes programas con punteros? Explique su respuesta**

**1.**

```
int* punt;
int x=7;
int y = 5;
punt=&x;
*punt = 4;
printf("%d %d", x, y);
```

**2.**

```
int* punt;
int x=7;
int y = 5;
punt=&x;
x = 4;
punt=&y;
printf("%d %d", *punt, x);
```

**3.**

```
int* punt, i;
int x[]={1, 2, 3, 4, 5};
punt = x;
*punt = 9;
for(i=0; i<5; i++){
   printf("%d", x[i]);
}
```

**4.**

```
int* punt, i;
int x[]={1, 2, 3, 4, 5};
punt = x;
*(punt+2) = 9;
*(x+3) = 7;
punt[1]=11;
for(i=0; i<5; i++){
   printf("%d", *(punt+i));
}
```

**5.**

```
int *punt,i;
int x[5]={1,2,3,4,5};
punt=&x[0]+3;
*(punt-2)=9;
punt--;
*(punt)=7 ;
punt[1]=11 ;
punt=x;
for(i=0;i<5;i++)
printf("%d,",punt[i]));
```

**6.**

```
int main() {
   int v[4] = { 2,4,5,7}, a, *p;
   p = v+2;
   p--;
   a = *p + *(p+1) + *(v+1) + p[2];
   printf("%d", a);
   return 1;
}
```

**7.**

```
void suma_dos(int* x, int* y, int z){
   *x = *x + 2;
   *y = *y + 5;
   z = z + 4;
}
int main() {
   int x, y, z;
   x = 3;
   y = 10;
   z = 15;
   suma_dos(&x, &y, z);
   printf("%d %d %d", x, y, z);
   return 1;
}
```

**8.**

```
void ingreseCadena(char* c){
   gets(c);
}
int main() {
   char* cadena = malloc(sizeof(char)* 10);
   gets(cadena);
   printf("%s\n", cadena);
   ingreseCadena(cadena);
   printf("%s", cadena);
   return 1;
}
```

**9.**

```c
#include <stdio.h>
#include <string.h>
int *fun0(int x) {
   return &x;
}
int main(int argc, char *argv[])
{
   int *ptr = NULL;
   ptr = fun0(2);
   printf("%d\n", *ptr);
   printf("%d\n", *ptr);
   return 0;
}
```

**Analice los programas dados a continuación y, escriba todos los errores en el manejo de memoria que considere que se están realizando, explique cuál sería la salida por pantalla del programa.**

**10.**

```c
#include<stdio.h>
int main() {
   char textoA[30] = "Agarrate Catalina";
   char textoB[30] = "El Cuarteto de Nos";
   char* p= textoA;
   char* q= textoB;
   printf("textoA: %s\ntextoB: %s\n", textoA, textoB);
   while(*p++ = *q++);
   printf("while(*p++ = *q++);\n");
   printf("textoA: %s\ntextoB: %s\n", textoA, textoB);
   return 1;
}
```

**11.**

```c
int main() {
   int array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   int* ptr;
   ptr = array;
   printf("array[0]>%d; *ptr: %d\n", array[0], *ptr);
   printf("array[1]>%d; *(ptr+1): %d\n", array[1], *(ptr+1));
   ptr++;
   printf("ptr++; *ptr: %d\n", *ptr);
   return 1;
}
```

**Para cada uno de los programas que se listan a continuación, elija la salida que supone que ocurrirá al ejecutarlo, luego compruebe si estaba en lo cierto. ¿Por qué se obtuvo cada resultado?**

**12.**

```
char* copiar_cadena(char* cad, int longitud){
  char* a = malloc(sizeof(char) * longitud);
  a = cad;
  return a;
}

int main(){
  char a[10] = "hola";
  char* b = copiar_cadena(a, 10);
  printf("%s %s\n", a, b);
  b[0] = 's';
  printf("%s %s\n", a, b);
}
```

Resultado:

**a** hola hola
hola hola

**b** hola ∗basura∗
hola ∗basura∗

**c** Segmentation fault

**d** hola hola
hola sola

**e** hola hola
sola sola

**f** Ninguna de las anteriores

**13.**

```
void foo(int* a){
  a = NULL;
}

int main(){
  int a[67];
  a[0] = 123;
  printf("%d\n", a[0]);
  foo(a);
  printf("%d\n", a[0]);
}
```

Resultado:

**a** 123
Segmentation fault

**b** 123
123

**c** 123
∗basura∗

**d** Segmentation fault

**e** Ninguna de las anteriores

**14.**

```
int main(){
  char *ptr = "hola mundo";
  ptr[0] = 's';
  printf("%s\n",ptr);
}
```

Resultado:

**a** hola mundo

**b** sola mundo

**c** Segmentation fault

**d** Ninguna de las anteriores

**Considere el código y luego explique si las expresiones, que se encuentran a continuación, son correctas; en caso de serlo, dé el valor de las mismas.**

**15.**

```
struct pack3 {
   int a;
};
struct pack2 {
   int b;
   struct pack3 *next;
};
struct pack1 {
   int c;
   struct pack2 *next;
};
int main(){
   struct pack1 data1, *dataPtr;
   struct pack2 data2;
   struct pack3 data3;
   data1.c = 30;
   data2.b = 20;
   data3.a = 10;
   dataPtr = &data1;
   data1.next = &data2;
   data2.next = &data3;
   return 1;
}
```

| Expresion | Correcta | Valor |
|---|---|---|
| data1.c | | 30 |
| dataPtr->c | | 30 |
| dataPtr.c | la anterior | |
| data1.next->b | | 20 |
| dataPtr->next->b | | 20 |
| dataPtr.next.b | anterior | |
| dataPtr->next.b | anterior | |
| (*(dataPtr->next)).b | | 20 |
| data1.next->next->a | | 10 |
| dataPtr->next->next.a | siguiente | |
| dataPtr->next->next->a | | 10 |
| dataPtr->next->a | no existe a | |
| dataPtr->next->next->b | no existe b | |