

PROGRAMACION AVANZADA - Taller2-1

- 1) ¿Por qué las siguientes instrucciones C/C++ levantarían un error en ejecución?:

```
int *p = NULL;  
*p = 100;
```

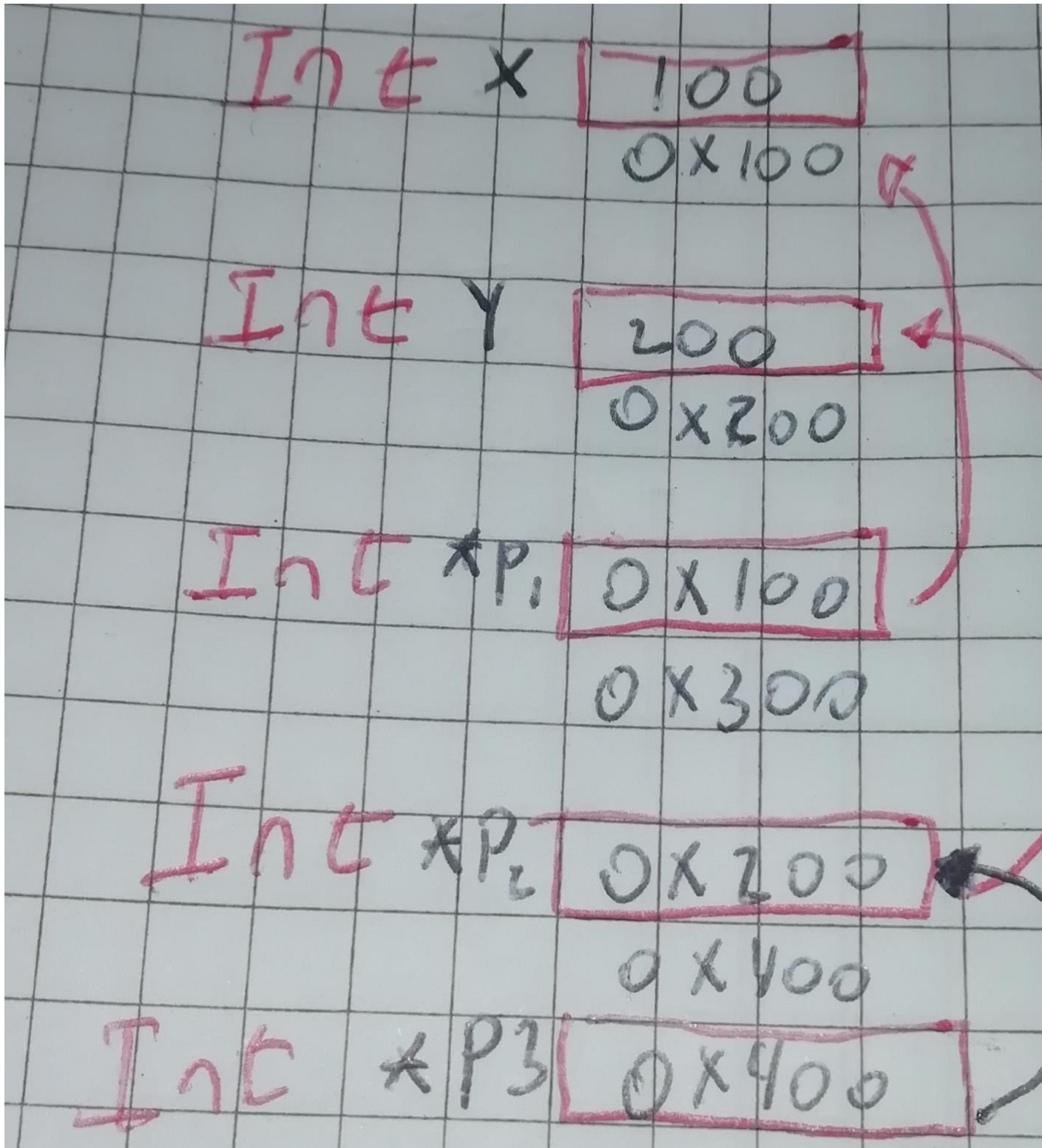
Por que p es un apuntador por lo tanto solo admite direcciones de memoria.

- 2) Establezca la instrucción que pondría al apuntador '*p' apuntar a la variable 'a':

```
int a = 100;  
int *p = NULL;  
  
p=&a;
```

- 3) Realice la respectiva representación gráfica (*como se hizo en la sesión de clase*) de la siguiente porción de código C/C++:

```
13      int x = 100;  
14      int y = 200;  
15      int *p1 = NULL;  
16      int *p2 = NULL;  
17      int **p3 = NULL;  
18  
19      p1 = &x;  
20      p2 = &y;  
21      p3 = &p2;
```



- 4) Para el código del Ítem (3), realice la instrucción que corresponda, tal que cambie el valor de 'y', desde '**p3', asignándole ahora a 'y' el valor apuntado por '*p1'.

```
**p3=*p1;
```

- 5) Para el código del Ítem (3), realice la instrucción que corresponda, tal que `***p3` ahora apunte al valor de `'x'`.

```
p3=&p1;
```

- 6) Para el código del Ítem (3), es válido pretender que `***p3` apunte **DIRECTAMENTE** a `'x'` o a `'y'`, ¿por qué?
Por que

Por que un apuntador de nivel 2 o mayor no puede apuntar directamente a una variable, si no que tiene que apuntar a un apuntador de nivel inferior

- 7) Para el código del Ítem (3), implemente las sgtes instrucciones:

Cree un apuntador `*p4`
Que `*p4` apunte también a `'x'`
Cree un apuntador `***p5`
Que `***p5` apunte también a `*p4`

Modifique el valor (el dato) apuntado indirectamente de `***p3` con el de `***p5`
Muestre el valor (el dato) apuntado indirectamente de `***p3` y el de `***p5`

```
int x=100;
int y=200;

int *p1=NULL;
int *p2=NULL;
int **p3=NULL;
int *p4=NULL;
int ***p5=NULL;
p4=&x;
p5=&p4;

p1=&x;
p2=&y;
p3=&p2;
**p3=***p5;
cout<<***p3<<endl;
cout<<***p5<<endl;
```

- 8) El siguiente código C/C++ por medio del apuntador `*p` mostrara los valores del arreglo

'arr', 1° la posición 1, luego el valor de la posición 0:

```

13     int arr[] = { 10, 20 };
14     int *p = NULL;
15
16     p = arr;
17
18     cout<<*(p+1)<<endl;
19     cout<<*p<<endl;

```

Muestre la suma de las posiciones del arreglo 'arr' usando el apuntador '*p' (por notación de apuntadores).

```
cout<<* (p) +* (p+1) <<endl;
```

- 9) Complete el siguiente código C/C++, defina correctamente el parámetro de la Línea#13 y como se debe pasar correctamente 'cadena' a f1 en la Línea#21, tal que sobre 'cadena' se haga un tratamiento por referencia y se muestre en la Línea#22: *Hola Companero piloso*

```

13 void f1(      ) {
14     strcat(p, " piloso");
15 }
16
17
18 int main() {
19     char cadena[32] = "Hola Companero";
20
21     f1(      );
22     cout<<cadena<<endl;
23
24     return 0;
25 }

```

```

#include <iostream>
#include<string.h>
using namespace std;
void f1(char *p) {
    strcat(p, "Piloso");
}
int main()
{
    char cadena[32]="Hola Compañero";
    f1(cadena);
    cout<<cadena<<endl;
    return 0;
}

```

- 10) Complete el siguiente código C/C++:

- Defina el 1° **parámetro** faltante tipo apuntador a función en la función '**calcularArea**'
(Linea#16 en el espacio en blanco antes de la 1° coma),
- Pase correctamente la variable que corresponda de tipo apuntador a función
(Linea#40 en el espacio en blanco antes de la 1° coma, no use valores literales, siempre variables)
- y pruebe el correcto funcionamiento del programa:

```

4   using namespace std;
5
6   enum TipoRectangulo { CUADRADO = 0, RECTANGULO = 1};
7
8   double obtenerAreaCuadrado(double base, double altura) {
9       return (pow(base, 2));
10  }
11
12  double obtenerAreaRectangulo(double base, double altura) {
13      return (base * altura);
14  }
15
16  double calcularArea(
17      double base, double altura, TipoRectangulo tiporectangulo) {
18      double resultado = 0;
19
20      resultado = (*rectangulogeneral)(base,altura);
21      switch(tiporectangulo) {
22          case CUADRADO:
23              //logica cuando sea un cuadrado
24              break;
25          case RECTANGULO:
26              //logica cuando sea un rectangulo
27              break;
28          default:
29              //logica cuando no se de ninguno de los anteriores casos
30              break;
31      }
32
33      return resultado;
34  }
35
36  int main() {
37      TipoRectangulo opcion = TipoRectangulo::CUADRADO; //opcion
38      double base = 5.0, altura = 10.0;
39      double(*wrapper[])(double,double) = {&obtenerAreaCuadrado,&obtenerAreaRectangulo};
40      double r = calcularArea(
41          ,base,altura,opcion);
42
43      cout<<"Area: "<<r<<endl;
44
45      return 0;
46  }

```

- ¿Qué ventaja da el uso de una **enum** (*enumeración*)?

proporciona una manera eficaz de definir un conjunto de constantes enteras con nombre que se pueden asignar a una variable.

Línea 16

```
double calcularArea(double (*rectangulogeneral)(double,
double), double base, double altura, TipoRectangulo
tiporectangulo);
```

Línea 40

```
double r = calcularArea( (*wrapper), base, altura, opcion);
```

11) Para el siguiente código C/C++, responda:

- ¿Que muestran las líneas 5 y 6?
- En notación de arreglo, ¿Como serían las coordenadas para mostrar el numero 70?

```

4 void f1(int (*p)[2][3]) {
5     cout<<*(*p)+8<<endl;
6     cout<<*(*p)+8)+5<<endl;
7 }
8
9 int main() {
10     int nums[2][2][3] = {
11         {
12             {10,20,30},
13             {40,50,60}
14         },
15         {
16             {70,80,90},
17             {100,110,120}
18         }
19     };
20
21     f1(nums);
22
23     return 0;
24 }
25

```

En la línea 5: imprime 8 posiciones después de la posición donde se encuentra el apuntador, en este caso 90.

En la línea 6: imprime 8 posiciones después de la posición donde se encuentra el apuntador y le suma 5 unidades, por eso imprimiría en este caso 95.

Para imprimir 70: `cout<<(*p)[1][3]<<endl;` o `cout<<(*p)[2][0]<<endl;` o `cout<<(*p)[0][6]<<endl;`

12) Con base al siguiente código C/C++:

- Realice la respectiva representación gráfica correcta, paso a paso de TODO lo que sucede en memoria.
- ¿Qué comportamiento tuvo el 1° caso?
- ¿Qué comportamiento tuvo el 2° caso?

```

4 void f1(int *p, int i) {
5     p = &i;
6 }
7
8 void f2(int **p, int i) {
9     *p = &i;
10 }
11
12 int main() {
13     int a = 100;
14     int *p = NULL;
15
16     p = &a;
17     f1(p, 30);
18     cout<<"1er Caso: "<<*p<<endl;
19     f2(&p, 30);
20     cout<<"2do Caso: "<<*p<<endl;
21
22     return 0;
23 }

```

F1 *P [80]

F2 **P [80]

Main

P [80]

F1 (P, 30) = 100 → a

F2 (P, 30) → 4714862 → 80

13) Con base al siguiente código C/C++:

- En teoría la Línea#14 debería mostrar: 100
¿Por qué no muestra 100?
Algo se debería hacer en la Línea#5, ¿Qué se debería hacer?

```
4 void f1(int **p) {  
5     int a = 100;  
6     cout<<&a<<endl;  
7     *p = &a;  
8 }  
9  
10 int main() {  
11     int *p = NULL;  
12     f1(&p);  
13     cout<<p<<endl;  
14     cout<<*p<<endl;  
15  
16     return 0;  
17 }
```

¿Por qué no muestra 100? No se puede muestra debido a que esta en el segmento de memoria stack (en la cima) donde si se imprime la variable, nos muestra es una dirección.

En la línea #5 incluir el valor static a la variable para mantenerla en el espacio estático de la memoria para permitir que se imprima el valor deseado.