Programación y Algoritmos

Tarea 3

Pablo Antonio Stack Sánchez

29/08/19

Problema (1)

1)

```
#include <stdio.h>

Struct mystruct {
        char a;
        char *b;
        char c;
};

int main () {
Printf ("\n Tamaño de la estructura : %Ld \n", sizeof(struct mystruct));
return 0;
}
```

2) Para el caso propuesto:

El tamaño de una estructura depende del tipo de variables que hay dentro. La forma en la que se alinea la memoria se realiza en multiplos del tamaño de variable más grande en la estructura.

El tamaño más grande del ejemplo está determinado por la variable char *b, Para el caso de una maquina de 64 bits es de 8 bytes, por lo tanto la memoria se alineará en "Paquetes" de 8 bytes.

Con la configuración que plantea el problema, tenemos los siguientes tamaños:

1 → char a

8 → char *b

1 → char c

Como se alinearán de 8 bytes en 8 bytes El compilador debe agregar los Paddings correspondientes; Por lo tanto quedaría:

El tamaño de la estructura será de 24 bytes

3)

Para minimizar el tamaño hay que juntar las variables "char a" y "char c" Para que sean alineados en un mismo paquete:

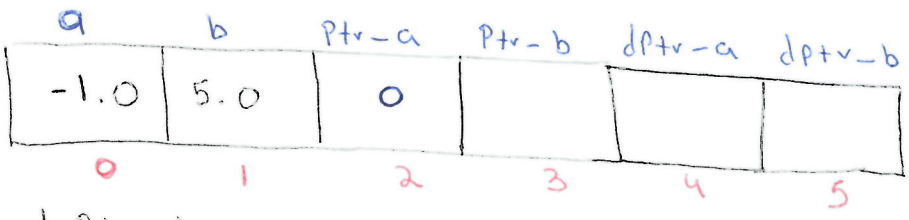
El tamaño mínimo serán 16 bytes

Ver tarea3_P1.c

Problema (2)

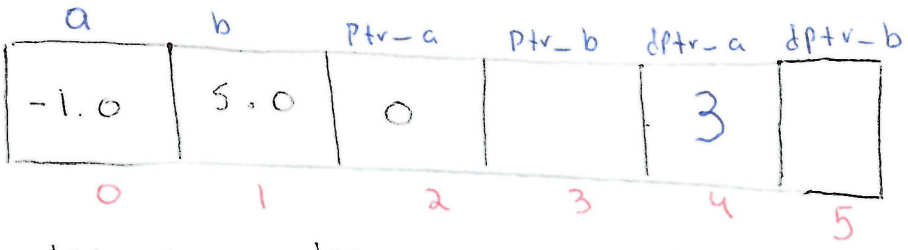
```
Ptr_a = &a;
```

a la variable Ptr_a se le pasa la dirección de memoria de la variable a.

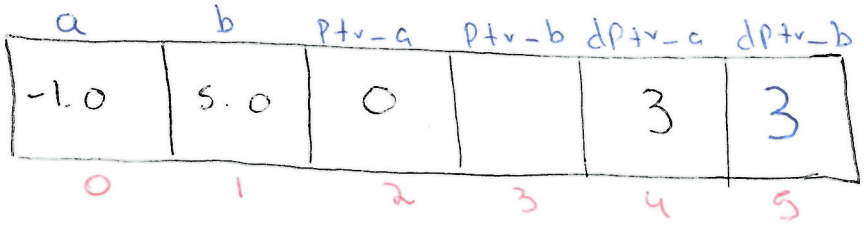



```
dPtr_a = &Ptr_b;
```

a la variable dPtr_a se le pasa la dirección de memoria de la variable Ptr_b:

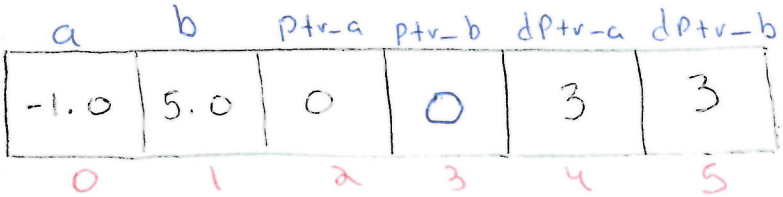


Al igualar dPtr_b con dPtr_a el contenido del segundo pasa al primero:




```
* dPtr_b = Ptr_a
```

El apuntador accede a la dirección de memoria que estaba guardada en dPtr_b, en este ejemplo sería "3" que corresponde a la variable Ptr_b y se le asigna el valor guardado en Ptr_a.




```
(** dPtr_a) = -3.0;
```

El doble apuntador primero accede a la dirección que estaba guardada en dPtr_a que es "3" que corresponde a Ptr_b. El segundo apuntador accede a la dirección guardada en Ptr_b que es "0" y corresponde a la variable a y se le asigna el valor de -3.

a	b	ptr_a	ptr_b	dptr_a	dptr_b
-3.0	5.0	0	0	3	3
0	1	2	3	4	5

al final si imprimos a tendrá el valor de -3.0.

Problema (3)

1)

El codigo hace un cast de un tipo flotante a un entero sin signo. y lo imprime como hexadecimal en la salida estandar de error

2)

```
#include <stdio.h>
int main () {
 double z = 2564;

 unsigned long int iz = * (unsigned long int*) &z;
 unsigned long int = corrimiento;

 For ( int i = size of (double) * sizeof (void*) -1 ; i >=0; i--) {
        corrimiento = iz >> i;
        corrimiento = corrimiento & 1;
       corrimiento == 1 ? Printf ("%u", 1) : Printf ("%u", 0);
 }

 return 0;
}
```

Problema 4

```
int ** Pascal (int n) {
    int suma = 0;
    int **X;
    X = (int **) malloc (n * sizeOF(int*));
    For (int i = 0; i < n; i++) {
        X[i] = (int *) malloc ((i+1) * sizeOF(int));
    }

    For (int i = 0; i < n; i++) {
        X[i][0] = 1;
        X[i][i] = 1;
        For (int j = 0; j < i; j++) {
            For (int k = j-1; k <= j; k++) {
                suma += X[i-1][k]
            }
            X[i][j] = suma;
            suma = 0;
            }
    }

    For (int i = 0; i < n; i++) {
        For (int j = 0; j <= i; j++) {
            Print F ("%d", X[i][j]);
        }
        Prin F ("\n");
    }
    return X;
}
```

Ver tarea3_p4.c

Este triangulo se podría mejorar, eliminando los unos de la primera columna y de la diagonal. además se podría almacenar unicamente la mitad del triangulo ya que la otra mitad se repite.

Problema (5)

Ver tarea 3 - P5.c