I2010 : langage C (TP3)

Avant de résoudre les exercices de cette fiche, faites le petit test CodeRunner « **TP3 Les pointeurs - Quiz** » sur Moodle.

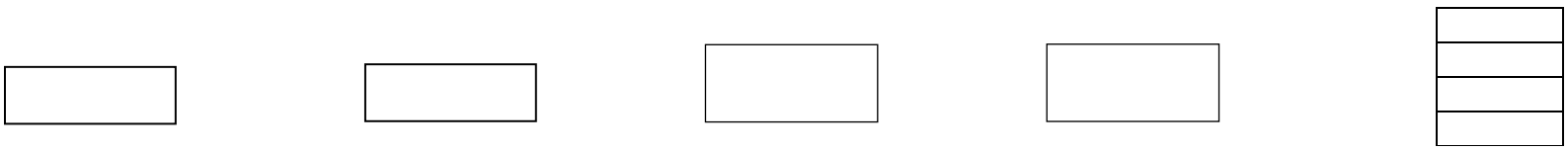
Pointeurs et gestion dynamique de la mémoire

1. Exercice de compréhension

Ci-dessous quelques déclarations et instructions, après chaque instruction, complétez le dessin pour montrer le contenu des variables.

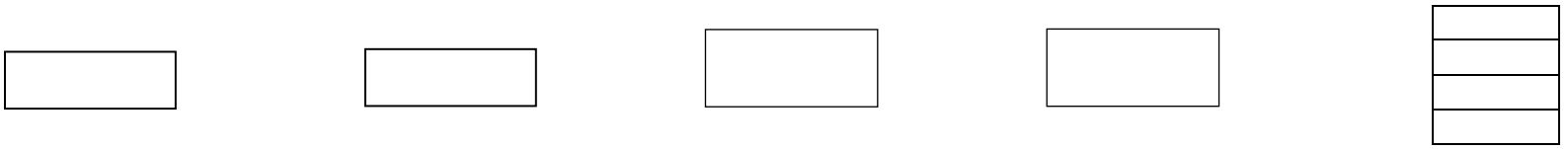
```
int main()
{
	int x = 1;
	int y = 1;
	int t[4] = {3, 4};
	int *ptr1, *ptr2;

	ptr1=&x;
	ptr2=t;
```

x **y** **PTR1** **PTR2** **t**

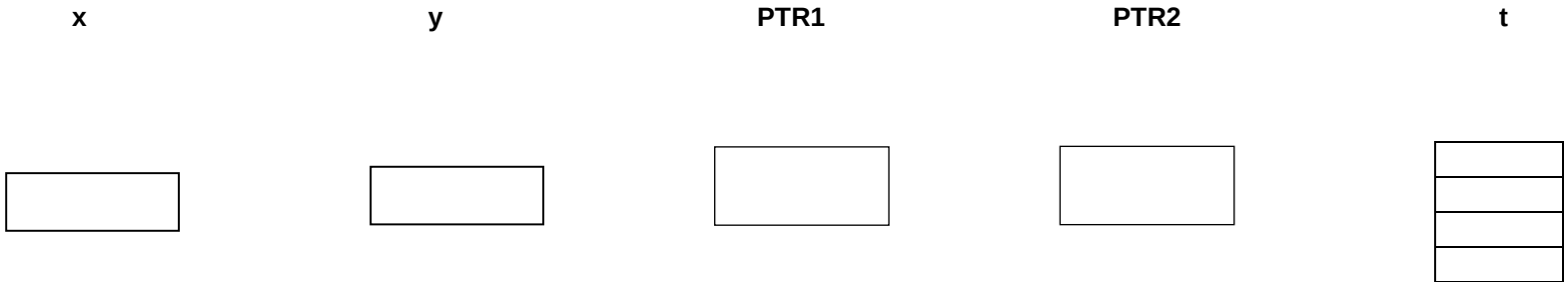


```
	(*ptr1)++;
```

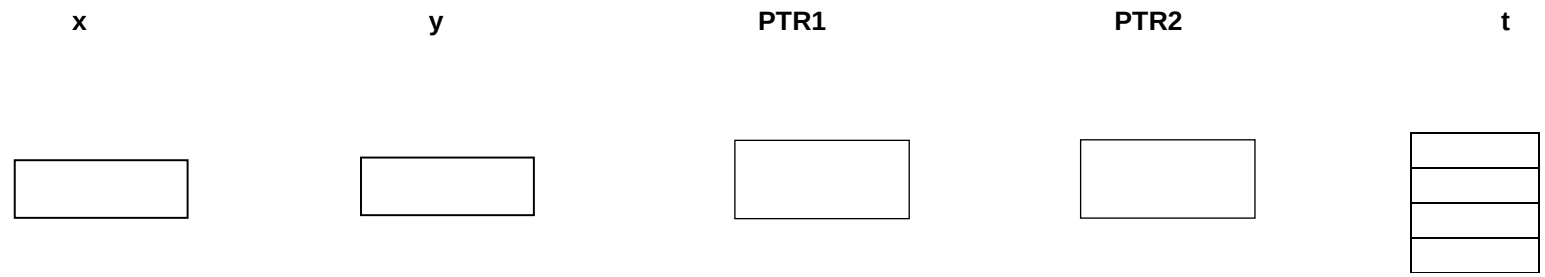
x **y** **PTR1** **PTR2** **t**



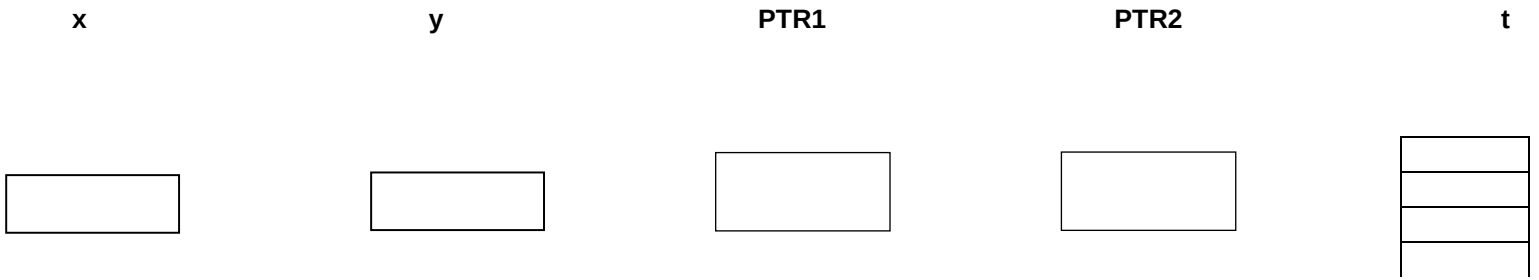
```
ptr2++;
```



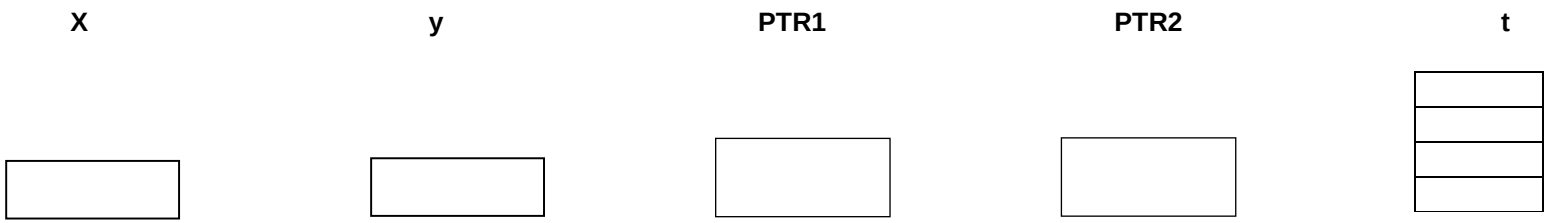

```
*(t+y) = *ptr1;
```



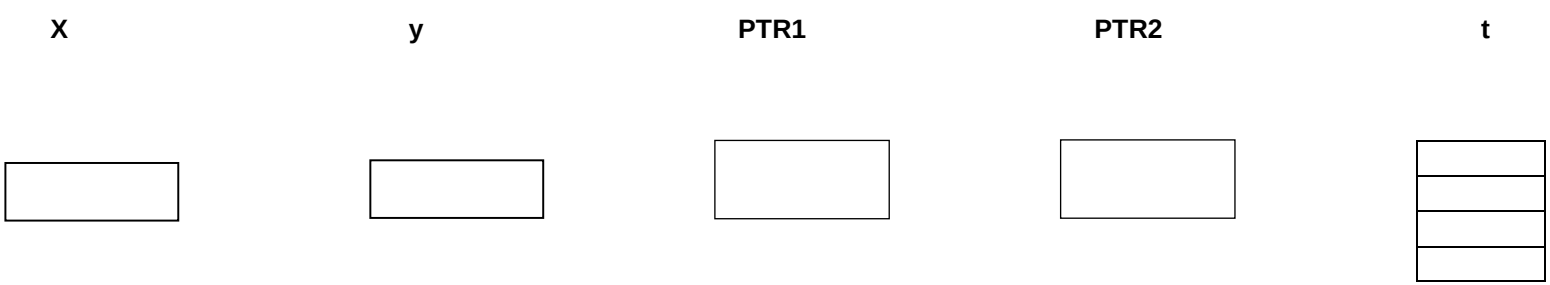

```
ptr1 = ptr2 + x;
```



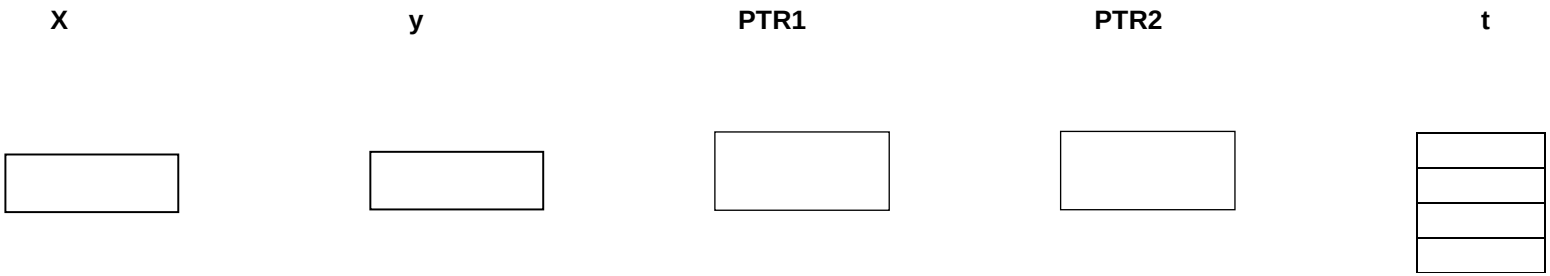
```
ptr1 = &(t[x+1]);
```




```
y = (*ptr1)++;
```




```
x = ptr1-t;
```




```
}
```

2. Allocation dynamique de tableaux à une dimension

a) Ecrivez un programme qui lit sur *stdin* :
   - un entier `n` qui représente le nombre de données ;
   - `n` entiers qui peuvent être soit positifs, nuls ou négatifs.

   Après avoir lu les données, le programme créera et affichera deux tableaux :
   - l'un contiendra la liste des entiers >=0 ;
   - l'autre la liste des entiers <0

   Exemple :

   Input
   ```
   5 -2 56 12 -3
   ```

   Output
   ```
   5 56 12
   -2 -3
   ```

   Utilisez la fonction *scanf*.

   Vous devez allouer dynamiquement les tableaux de sorte que tous les éléments soient assignés (i.e. taille logique = taille physique) et libérer la place qu'ils occupent après leur utilisation.

b) Après avoir traité les entrées et affiché le résultat, le programme redemandera un nouveau jeu de données à traiter tant que `n>0`.

3. Arithmétique des pointeurs

Pour rappel, lorsque vous avez un pointeur dans un tableau, vous pouvez passer à l'élément suivant en incrémentant simplement ce pointeur (via l'opérateur ++).

Passez d'une version indicée à une version pointeurs du programme 2 : modifiez-le afin de ne plus utiliser l'opérateur d'indexation **[ ]**.

Exercices d'observation et debugging

4. Utilisation du debugger *gdb*[1]

Prenez sur Moodle les programmes sources suivant :

1. *to_debug_stack_1.c*
2. *to_debug_stack_smashing.c*
3. *to_debug_segmentation_fault_1.c*
4. *to_debug_segmentation_fault_2.c*
5. *to_debug_stack_smashing_2.c*
6. *to_debug_doublette.c*

Certains de ces programmes comportent des fautes de style et/ou des erreurs de bonne gestion de la mémoire.

Compilez et exécutez-les dans cet ordre. Trouvez quel est le problème, et si c'est pertinent, comment le corriger. Mais vous ne devez **pas corriger ces programmes** !

Pour les 2 programmes avec *segmentation fault*, utilisez **gdb**.

[1] Pour les possesseurs de Mac, utilisez le débugger *lldb*.