Lancer deux fils d'exécution en parallèle en C

Pour gérer des fils d'exécution en C on utilise la bibliothèque `pthread`. Ainsi, on veillera à indiquer `include <pthread.h>` en tête du fichier `main.c`, et à le compiler avec l'option `-pthread` :

```
gcc -pthread main.c -o main
```

Les fils d'exécution sont des objets de type `pthread_t`. Ils doivent d'abord être déclarés puis sont lancés par la fonction `pthread_create` qui prend 4 arguments :

- l'adresse du fil d'exécution, de type `pthread_t*` donc ;
- une adresse qui ne nous est pas utile, on mettra donc `NULL` ;
- une fonction `void* todo(void* arg);`, qui est celle que le fil d'exécution doit exécuter ;
- un pointeur vers les arguments de type `void*`.

Il est donc nécessaire de formater les instructions à effectuer dans une fonction `void* todo(void* arg)`. Pour cela on crée d'abord une structure `struct arg_s` qui rassemble les données utiles à `todo` dans un même objet `a` dont l'adresse sera passée au fil d'exécution comme 4-ème argument de `pthread_create`. La fonction `todo` commence par transtyper son argument `arg` pour y reconnaître un pointeur de type `(struct arg_s)*`. Les calculs peuvent donc être codés en accédant aux données par `a->champ`. Enfin le résultat ne peut être retourné en sortie de `todo`, mais doit être enregistré dans un objet existant hors de la fonction (on ajoute parfois son adresse dans la structure `arg_s`). Autrement dit la fonction `todo` doit non seulement prendre ses entrées à travers un unique pointeur, mais doit aussi agir par effet de bord uniquement.

On attend que plusieurs fils d'exécution aient terminé leur exécution grâce à la fonction `pthread_join` qui prend 2 arguments :

- le processus, de type `pthread_t` donc ;
- une adresse qui ne nous est pas utile, on mettra donc `NULL`.

```c
#include <pthread.h>
#include <assert.h>

struct args_s {
  int nb;
  int* res;
};
typedef struct args_s arg_carre;

void* au_carre(void* args) {
  arg_carre* a = (arg_carre*) args;
  *(a->res) = a->nb * a->nb;
  return NULL;
}

int main(){
  int resA, resB;
  arg_carre argsA = {2, &resA};
  arg_carre argsB = {9, &resB};
  pthread_t pA, pB;
  pthread_create(&pA, NULL, au_carre, &argsA);
  pthread_create(&pB, NULL, au_carre, &argsB);
  pthread_join(pA, NULL);
  pthread_join(pB, NULL);
  assert((resA == 4) && (resB == 81));
  return 0;
}
```