TD 1 - C - Classe de stockage, types, pointeurs et manipulation mémoire

Hugo Bolloré[1]

[1]hugo.bollore@uvsq.fr

Exercice 1.1 Classes de stockage

Question 1.1.1 Statique et registre

Copiez le code suivant et expliquez les résultats affichés par le programme.

```c
#include <stdio.h>

void storageClassesTest() {
  // static_val is static (scope: within block, life: till end of program)
  static int static_val = 1;

  // register_val is a register variable (scope: within block, life: till end of block)
  register int register_val = 1;

  // incrementing variables
  static_val++;
  register_val++;

  printf("Static value: %d \n", static_val);
  printf("Register value: %d \n", register_val);
}

void main() {
  storageClassesTest();
  storageClassesTest();
}
```

Que fait la classe de stockage *register* ?

Question 1.1.2 Chaînes de caractères et classe de stockage automatique

Copiez et compilez le code suivant :

```c
char * obi() {
    return "Hello";
}
char * one() {
   char str[] = "there";
   return str;
}
```

Expliquez pourquoi le compilateur n'accepte pas le code de la fonction *one* alors qu'il accepte celui de la fonction *obi*. Pour cela, essayez de retrouver quelle est la classe de stockage de tous les éléments dans les deux fonctions.

Question 1.1.3 Externe

Implémentez un exemple montrant l'utilisation de la classe de stockage externe (*extern*). Pour cela vous devrez créer les fichiers suivants :

- Un fichier answer.h qui contiendra la *déclaration* de la variable *answer*,
- Un fichier answer.c qui contiendra la *définition* de la variable *answer*,
- Un fichier joke.c qui affichera le texte "Why do computer scientists get confused between Halloween and Christmas ?" et appelera ensuite la fonction suivante :

```
printf(answer, 8*sizeof(int) - 1, 8*(sizeof(short) + sizeof(char)) + 1);
```

La variable *answer* sera une chaîne de caractères qui devra contenir "Because Oct %d = Dec %d".

Bonus : expliquez la réponse.

Exercice 1.2 Allocation mémoire et pointeurs

Question 1.2.1 *malloc* et *calloc*

Lisez la documentation des fonctions C *malloc* et *calloc* et implémentez un programme qui répond aux contraintes suivantes :

- Il doit prendre en paramètre le nombre d'éléments à allouer (ce seront des entiers non signés)

  par exemple : *./my_awesome_program 42*,
- Il doit allouer dynamiquement un premier tableau qui doit pouvoir contenir ces éléments en utilisant la fonction *malloc*,
- Il doit allouer dynamiquement un deuxième tableau qui doit pouvoir contenir ces éléments en utilisant la fonction *calloc*,
- Il n'initialise PAS les valeurs dans ces deux tableaux,
- Il doit afficher l'adresse et la valeur de chacun des éléments de ces deux tableaux.

Exercice 1.3 Arithmétique de pointeurs

Question 1.3.1 Parcours d'un tableau

Reprennez le code de la question 1.2.1 et ajoutez une initialisation des valeurs des deux tableaux en utilisant uniquement de l'arithmétique de pointeurs.

Question 1.3.2 Void

Commentez l'initialisation des valeurs que vous avez ajouté précedemment puis changez la déclaration de vos tableaux pour que ce ne soit plus des pointeurs sur des entiers mais des pointeurs sur *void*. Vous aurez besoin de *cast* vos tableaux en *int ** pour afficher leurs valeurs.

Question 1.3.3 Arithmétique de pointeurs et void

Décommentez l'initialisation des valeurs des tableaux. Que remarquez vous ? Que faut-il changer pour que l'initialisation fonctionne de nouveau ?