Feuille d'exercices n°3 - Gestion de la mémoire en C

Notions abordées

- passage par référence (via un pointeur) vs passage par valeur
- variables locales aux fonctions
- représentation de l'évolution de la pile à l'exécution
- intervalles d'entiers : cardinal, test de vacuité, union, intersection, somme et différence

Exercice 0 Pour s'échauffer

Question 1 ✍

Donner l'affichage que produit le code ci-dessous.

```c
int a = 10;
int* pa = &a;
int b = 103;
int* pb = &b;

printf("a == b : %d\n",a == b);
printf("pa == pb : %d\n",pa == pb);
printf("----\n");

b = b/a;
int* y = &a;
int* z = &b;

printf("a == b : %d\n",a == b);
printf("pa == pb : %d\n",pa == pb);
printf("pa[0] == pb[0] : %d\n",pa[0] == pb[0]);
printf("----\n");

printf("y == z : %d\n",y == z);
printf("y == pa : %d\n",y == pa);
printf("z == pb : %d\n",z == pb);
printf("----\n");
```

Question 2 ✍

Donner l'affichage que produit le code ci-dessous.

```
  char* s = "hello";
  char** pp1 = &s;
  char* p1 = s;
  char c1 = s[4];
  char* ac1 = &c1;

  printf("s+2 == s+3 : %d\n",s+2 == s+3);
  printf("s[2] == s[3] : %d\n",s[2] == s[3]);
  printf("----\n");

  s = "olé";
  char** pp2 = &s;
  char* p2 = s;
  char c2 = s[0];
  char* ac2 = &c2;

  printf("p1 == p2 : %d\n",p1 == p2);
  printf("pp1 == pp2 : %d\n",pp1 == pp2);
  printf("c1 == c2 : %d\n",c1 == c2);
  printf("ac1 == ac2 : %d\n",ac1 == ac2);
  printf("*ac1 == ac2[0] : %d\n",*ac1 == ac2[0]);
```

Évolution de la pile à l'exécution

Exercice 1 Premiers exemples

Question 1 ✍

Représenter l'état de la pile à chaque étape dans le programme ci-dessous.

```
void  f (int* p1, int a){
  int var_loc = 3;
  *p1 = var_loc + a;
  a = var_loc + 1;
}

int main(){
  int v1 = 10;
  int v2 = 3;
  f(&v2, v1);
  return 0;
}
```

Question 2 ✍
Représenter l'état de la pile à chaque étape dans le programme ci-dessous.

```
1  int f(int i1, int i2){
2    int i3;
3    i3 = i1 + i2;
4    i2 = i3;
5    return i3;
6  }
7
8  int main(){
9    int i1 = 1;
10   int i2 = 2;
11   int res;
12   res = f(i1,i2);
13   return 0;
14 }
```

Question 3 ✍
Pourquoi l'instruction ligne 4 dans la fonction `f` définie ci-dessus est-elle inutile ? Corriger la fonction pour que cette dernière affectation devienne utile, c'est-à-dire observable au delà de l'appel de fonction. Modifier en conséquence la fonction `main`, puis représenter à nouveau les états successifs de la pile.

Exercice 2 Mystère

Question 1 ✍
Dans la fonction `mystere` définie ci-contre, dire ce qui est passé par valeur et ce qui est passé par référence.

```
1  void mystere ( int* pa, int a){
2    *pa = *pa * a;
3    *pa = *pa * a;
4    a = a * a;
5  }
```

Question 2 ✍
Donner une description de la fonction `mystere` définie ci-dessus.

Question 3 ✍
Quel affichage réalise le programme ci-dessous. Représenter l'état de la pile à chaque étape.

```
1  int main(){
2    int a = 2;
3    int* pa = &a;
4    printf("0--- a = %d\n", a);
5    mystere(pa,a);
6    printf("1--- a = %d\n", a);
7    mystere(pa,a);
8    printf("2--- a = %d\n", a);
9    return 0;
10 }
```

Question 4 ✍
Si on continuait à appeler `mystere(pa,a)` puis à afficher `a`, quelle valeur serait affichée après le k-ième appel. Justifier.

Exercice 3 Addition bit à bit

Dans cet exercice on implémente l'algorithme d'addition des entiers naturels écrits en binaire, à partir de fonctions élémentaires qui réalisent l'addition de 2 bits, puis l'addition de 2 bits et une retenue (on précise plus bas ce que l'on entend exactement par là). On représentera les chiffres utilisés en binaire (0 et 1) par des booléens (`false` et `true` respectivement), ainsi les nombres seront des tableaux de booléens.

Question 1 ✍

On cherche d'abord à additionner 2 bits, c'est-à-dire à savoir quel chiffre inscrire dans le résultat, et s'il y a une retenue. Définir une fonction `demi_add` réalisant cette opération *Comme on veut retourner deux valeurs, on va ici passer deux pointeurs pour enregistrer ces valeurs.*

Question 2 ✍

Représenter l'évolution de la pile à l'exécution des programmes suivants.

```
int main() {
  bool x = true;
  bool y = false;
  bool res;
  bool ret;
  demi_add(x, y, &res,
   ↪  &ret);
}
```

```
int main() {
  bool x = true;
  bool y = false;
  bool res;
  demi_add(x, y, &res,
   ↪  &res);
}
```

```
int main() {
  bool x = true;
  bool y = false;
  demi_add(x, y, &x,
   ↪  &x);
}
```

Question 3 ✍

On cherche maintenant à additionner 2 bits en tenant compte d'une éventuelle retenue. Sachant qu'une fois que la retenue est prise en compte dans l'addition des bits suivants elle n'est plus utile, on stockera la nouvelle retenue à la place que la retenue passée en argument. Définir une fonction `add` réalisant cette opération. *Votre fonction doit utiliser la fonction précédente et des opérateurs logiques.*

Question 4 ✍

On cherche enfin, à additionner 2 nombres écrits en binaire. Définir une fonction `addition` réalisant cette opération. *Votre fonction doit utiliser la fonction précédente et des opérateurs logiques, et stocker le nombre résultat dans un tableau.*

Gestion dynamique de la mémoire

Exercice 4 Tables dynamiques - partie 1 : ajouts

On modélise dans cet exercice des tableaux d'entiers strictement positifs dont la taille peut augmenter. On utilisera un tableau contenant la valeur 0 pour indiquer une case vide. Les autres valeurs du tableau, strictement positives, qui sont les valeurs significatives, sont toutes placées en début de tableau. Autrement dit les cases vides sont toutes placées à la fin.
Comme il est utile pour manipuler un tableau d'avoir sa taille, on veut associer au tableau des éléments sa taille. De plus on aimerait aussi avoir accès en temps constant au nombre d'éléments significatifs (*i.e.* non nuls) du tableau.
On appellera table un tel objet (*i.e.* le tableau, sa longueur et son nombre d'éléments significatifs). On parlera d'éléments de la table pour désigner les éléments du tableau correspondant.

Question 1 ✍
Déclarer la structure `s_table` permettant de modéliser les tables décrites ci-dessus, et donner une représentation graphique d'une instance de cette structure.
Définir ensuite le type `table` par `typedef struct s_table table;`.

Question 2 ✍
Définir une fonction `cree_table_vide` qui prend en argument un entier strictement positif `lg` et qui retourne une table vide de longueur `lg`.

Question 3 ✍
Définir une fonction `est_valide` qui prend en argument une table et qui retourne si elle est valide au sens où ses champs ne sont pas aberrants et cohérents entre eux. En particulier on vérifiera que certains éléments de la table sont strictement positifs, et que d'autres sont nuls.

Question 4 ✍
Définir une fonction `est_plein` qui prend en argument une table et qui retourne si elle est pleine.

Question 5 ✍
Définir une fonction `affiche` qui prend en argument une table et qui affiche d'abord ses éléments significatifs, puis qui affiche le nombre de cases vides.

Question 6 ✍
Définir une fonction `copie_double` qui copie les éléments d'une table dans un tableau deux fois plus grand. *On veillera à ce que la table reste valide, et aussi à libérer l'espace mémoire qui n'est plus utile.*

Question 7 ✍
Définir une fonction `ajoute` qui ajoute un entier strictement positif à la table, et qui au besoin (*i.e.* s'il n'y a plus de cases vides) copie les éléments du tableau dans un tableau deux fois plus grand.

Question 8 ✍
Proposer un programme qui crée une table vide, puis y ajoute successivement les entiers de 1 à 7, et affiche la table à chaque étape.

Rien à voir (suite au DS)

Exercice 5 Intervalles d'entiers

On travaille dans cet exercice sur des intervalles entiers (et donc intervalle signifie à chaque fois intervalle d'entiers). On rappelle que pour $(x, y) \in \mathbb{R}^2$, on note $[x..y]$ l'ensemble des nombres entiers supérieurs à x et inférieurs à y (au sens large), et que cet ensemble peut être vide.
On appelle ici **intersection** de deux intervalles $\mathcal{I}$ et $\mathcal{J}$ le plus grand intervalle contenant uniquement des éléments à la fois dans $\mathcal{I}$ et dans $\mathcal{J}$. On appelle ici **union** de deux intervalles $\mathcal{I}$ et $\mathcal{J}$ le plus petit intervalle contenant tous les éléments de $\mathcal{I}$ et tous ceux de $\mathcal{J}$. On note que l'union et l'intersection sont donc vues ici comme des opérations binaires sur les intervalles d'entiers : l'union de deux intervalles est un intervalle, et l'intersection de deux intervalles est un intervalle.

Question 1 ✍
Soit $(a, b) \in \mathbb{Z}^2$. À quelle condition l'intervalle $[a..b]$ est-il non vide ?

Question 2 ✍
Soit $(a, b, c, d) \in \mathbb{Z}^4$. En supposant que $[a..b] \cap [c..d]$ est non vide, quelles sont les bornes de l'intervalle $[a..b] \cap [c..d]$? Sans cette supposition, comment décrire cet intervalle.

Question 3 ✍
Soit $(a, b, c, d) \in \mathbb{Z}^4$. À quelle condition l'intervalle $[a..b] \cap [c..d]$ est-il vide ?

Question 4 ✍
Soit $(a, b) \in \mathbb{Z}^2$. En supposant que $[a..b]$ est non vide, quel est le cardinal de $[a..b]$? Sans cette supposition, comment exprimer le cardinal de $[a..b]$?

Question 5 ✍
Soit $(a, b, c, d) \in \mathbb{Z}^4$. En supposant que $[a..b]$ et $[c..d]$ sont non vides, quelles sont les bornes de l'intervalle $[a..b] \cup [c..d]$? Sans cette supposition, comment décrire cet intervalle.

Pour A et B deux sous-ensembles d'un ensemble muni d'une addition, on appelle **somme de Minkowski** de ces ensembles l'ensemble noté $A + B$ défini par $A + B = \{a+b \mid a \in A, b \in B\}$.
Dans cet exercice on utilise cette définition dans le cas particulier où les ensembles sont des intervalles d'entiers, ce qui a bien du sens car $\mathbb{Z}$ est muni d'une addition.

Question 6 ✍
Soit $(a, b, c, d) \in \mathbb{Z}^4$. En supposant que $[a..b]$ et $[c..d]$ sont non vides, montrer que $[a..b] + [c..d]$ est un intervalle et donner ses bornes?

Question 7 ✍
Soit $(a, b, c, d) \in \mathbb{Z}^4$. En supposant que $[a..b]$ et $[c..d]$ sont non vides, donner le cardinal de $[a..b]+[c..d]$? D'une manière générale, quelle est le cardinal de la somme de deux intervalles de cardinaux respectifs n et m quand $n > 0$ et $m > 0$?

Question 8 ✍
Peut-on donner le cardinal de l'union ou l'intersection de deux intervalles en fonction de leurs cardinaux respectifs n et m quand $n > 0$ et $m > 0$?