Tableaux, la théorie

1 Les tableaux en mémoire

Exercice 1. État de la mémoire sur un tableau simple.

Déroulez le code ci-dessous et complétez l'état de la mémoire au fur et à mesure de l'exécution du code.

```
int i = 0;
char tab[4] = {'a', 'b'};
for (i=0; i<4; i++)
  tab[i] = (char) i;
```

Adresse	Nom de var.	Valeur de var.
0x2D30		
0x2D31		
0x2D32		
0x2D33		
0x2D34		
0x2D35		
0x2D36		
0x2D37		
0x2D38		
0x2D39		
0x2D3A		
0x2D3B		
0x2D3C		
0x2D3D		
0x2D3E		
0x2D3F		
0x2D40		
0x2D41		
0x2D42		
0x2D43		

Exercice 2. État de la mémoire : tableau et appel de fonction.

Déroulez le code ci-dessous et complétez l'état de la mémoire au fur et à mesure de l'exécution du code.

```
void func (short tab[], short n)
{
  tab[0] = n;
}

...

short t[4] = {1,3,3,7};
short i = 0;
func (t, i);
```

Adresse	Nom de var.	Valeur de var.
0x2D30		
0x2D31		
0x2D32		
0x2D33		
0x2D34		
0x2D35		
0x2D36		
0x2D37		
0x2D38		
0x2D39		
0x2D3A		
0x2D3B		
0x2D3C		
0x2D3D		
0x2D3E		
0x2D3F		
0x2D40		
0x2D41		
0x2D42		
0x2D43		

Exercice 3. État de la mémoire et tableau 1D.

La question 3 de l'exercice 3 du TD précédent vous demandait de rédiger une fonction calculant la moyenne des éléments. Le code ci-dessous présente la correction de cet exercice manipulant un tableau de `short`. Remplissez l'état de la mémoire à la fin de l'exécution du code ci-dessous.

```
float moy (short tab[], int size)
{
  float res = 0.f;
  int i;

  for (i=0; i<size; i++)
    res += (float)tab[i];

  return res/size ;
}

int main()
{
  short t[4] = {1,2,3,4};
  float res;

  res = moy (t, 4);
  printf("moyenne = %f\n", res);

  return 0;
}
```

Adresse	Nom var.	Valeur var.
0x120		
0x121		
0x122		
0x123		
0x124		
0x125		
0x126		
0x127		
0x128		
0x129		
0x12A		
0x12B		
0x12C		
0x12D		
0x12E		
0x12F		
0x130		
0x131		
0x132		
0x133		
0x134		
0x135		
0x136		
0x137		
0x138		
0x139		
0x13A		
0x13B		
0x13C		
0x13D		
0x13E		
0x13F		

Exercice 4. État de la mémoire : Tableau 2D.

Déroulez le code ci-dessous et complétez l'état de la mémoire au fur et à mesure de l'exécution du code.

```
char func (char mat[][2], int n)
{
  int i;
  for (i=0; i<n-1; i++)
    mat[i][0] = 'i';
}
...
char mots[3][2] = {{'p', 'b'},
                   {'f', 'j'},
                   {'a', 'u'}};
func (mots, 3);
```

Adresse	Nom var.	Valeur var.
0x2D30		
0x2D31		
0x2D32		
0x2D33		
0x2D34		
0x2D35		
0x2D36		
0x2D37		
0x2D38		
0x2D39		
0x2D3A		
0x2D3B		
0x2D3C		
0x2D3D		
0x2D3E		
0x2D3F		
0x2D40		
0x2D41		
0x2D42		
0x2D43		

Exercice 5. État de la mémoire : tableau 2D et appel de fonction.
Remplissez l'état de la mémoire à la fin de l'exécution du code ci-dessous.

```
void sum(short mat[][3], int size)
{
  int i, j;
  short tmp= 0;

  for (i=0; i<size; i++)
  {
    for (j=0; j<3; j++)
    {
      tmp += mat[i][j];
      mat[i][j] = tmp;
    }
  }
}

int main()
{
  short m[2][3] = {{11, 12, 13},
                   {21, 22, 23}};
  sum(m, 2);
  return 0;
}
```

Adresse	Nom var.	Valeur var.
0x120		
0x121		
0x122		
0x123		
0x124		
0x125		
0x126		
0x127		
0x128		
0x129		
0x12A		
0x12B		
0x12C		
0x12D		
0x12E		
0x12F		
0x130		
0x131		
0x132		
0x133		
0x134		
0x135		
0x136		
0x137		
0x138		
0x139		
0x13A		
0x13B		
0x13C		
0x13D		
0x13E		
0x13F		

2 Pointeurs et décalage d'adresse

Exercice 6. Décalage d'adresse.

L'objectif de cet exercice est de manipuler les adresses d'un tableau. Rédigez tout d'abord un code déclarant un tableau `tab` de 10 entiers dont la case `i` contient l'entier `i`.

1) Affichez l'adresse de votre tableau.
2) Affichez l'adresse de `tab[0]` et de `tab[1]`. Pas de surprise jusque là.
3) Déclarez un pointeur sur entier nommé `p` et pointant sur la première case de votre tableau.
4) Qu'affiche l'instruction `printf("p pointe sur %p\n", p);` ?
5) Qu'affiche l'instruction `printf("p+1 pointe sur %p\n", p+1);` ?
6) Qu'affiche l'instruction `printf("p+2 pointe sur %p\n", p+2);` ?
7) Déduisez-en une écriture équivalente de `p+i`.

Exercice 7. Le pouvoir `void *`.

Considérons le code suivant.

```
int main()
{
    void *p = &p;
    printf( "%p\n", p );
    printf( "%p\n",       (char*)p + 1 );
    printf( "%p\n",      (short*)p + 1 );
    printf( "%p\n",        (int*)p + 1 );
    printf( "%p\n",      (float*)p + 1 );
    printf( "%p\n", (long long*)p + 1 );
    printf( "%p\n",     (double*)p + 1 );
}
```

1) En supposant que l'adresse du pointeur `p` soit `0x2D30`, qu'affiche le code suivant ?
2) Vérifiez votre réponse en exécutant le code.
3) Que se passe-t-il si vous ajoutez la ligne `p++;` au code précédent ? Pourquoi ?

3 Chaines de caractères

Exercice 8. Cap' ou pas cap' ?.

Pour chacune des instructions suivantes, indiquez si le compilateur va l'accepter, et le cas échéant, si le code va fonctionner. Dans tous les cas, détaillez ce qu'il se passe. Vous vérifierez vos résultats en compilant et exécutant chacune de ces instructions.

1) `char str01[] = "hello";`
2) `char *str02 = NULL;`
3) `char str03[200] = "ibijau";`
4) `char str04[7] ={'patate'};`
5) `char *str05 = "1337";`
6) `char *str06 = "potoo"` puis `str06[0] = 'P';`
7) `char str07[] = "bird"` puis `str07[4] = '!';`

8) `char *str08 = "pasteque"` puis `str08 = "ananas";`
9) `char str09[] = "chaine"` puis `str09 = "maison";`

Exercice 9. Manipulations de chaines (Moodle).

Nous vous proposons les six fonctions suivantes. La consigne est simple : codez-en au moins quatre. Pour deux d'entre elles, utilisez l'arithmétique des pointeurs.

1) Rédigez la fonction `longueurChaine` permettant de renvoyer le nombre de caractères de la chaine passée en argument de la fonction.
2) Rédigez la fonction `nbVoyelles` renvoyant le nombre de voyelles présentes dans la chaine.
3) Rédigez la fonction `inverseCasse` qui inverse la casses des lettres (majuscules vers minuscules et inversement).
4) Rédigez la fonction `chaineVersEntier` prenant un chaine de caractères composée de chiffres et renvoyant l'entier associé. Si la chaine est trop longue, affichez une erreur.
5) Rédigez la fonction `nbOcc` prenant une chaine et un caractère et renvoyant le nombre de fois où le caractère est présent dans la chaine
6) Rédigez la fonction `compareChaine` prenant deux chaines et renvoyant 0 si les chaines sont identiques, 1 si la première est avant la seconde dans l'ordre lexicographique, -1 sinon.