INF203 - Exercices semaine 6

Retour sur TP5, chaînes de caractères, structures

Exercice 1 :

Au sujet des programmes `deborde_char` ... Que signifie cette histoire de débordement ?

1. Combien de valeurs distinctes peut-on coder avec 1, 2, 4, n bits ?
2. Si on code des entiers (non signés), à quel intervalle est-ce que ça correspond ?
3. Que se passe-t-il si on ajoute 1 à `0b111...1` ?
   Par combien multiplie-t-on le nombre de valeurs codables en écrivant un entier sur 1 bit de plus ?
   Et sur n bits de plus ?
4. Par combien est multiplié le temps d'exécution entre `deborde_char` et `deborde_short` ?
   Entre `deborde_short` et `deborde_int` ?
   Et si on prenait des entiers sur 64 bits (`long long`), ça prendrait combien de fois plus de temps que `deborde_int` ?
   Ça prendrait combien de temps environ ?

Solution :

1. 2^n
2. un entier étant codé sur 32 bits (sur turing), les chiffres vont de 0 à $2^{32} - 1 = 4294967295$ (attention au -1 vu que 0 est inclus)
3. le chiffre suivant vaut 0 ; avec un bit de plus, on double le nombre d'entiers codables ; avec n bits de plus, on multiplie par 2^n
4. entre `deborde_char` et `deborde_short`, on perd sur turing un rapport 2^8 ; entre `deborde_short` et `deborde_int`, il s'agit même de 2^{16}. Sur 64 bits, on perdrait un rapport 2^{32}. Sur turing, `deborde_int` prend un peu plus de 10 s. Pour des entiers sur 64 bits, ça prendrait donc plus de $2^{32} \times 10s$. Notons T ce temps.
   $T = 4 \times 2^{30} \times 10s \sim 4 \times 10^9 s = 4 \times 10^{10} s$
   1 jour= $86400s < 10^5 s$, donc $T > 4 \times 10^5$ jours.
   1 an = 365 jours $< 4 \times 10^2$ jours.
   Donc, $T > 10^3$ ans. Ça laisse le temps d'aller boire le café pendant que ça s'exécute.

Exercice 2 :

1. Écrire une fonction `ordonne` qui prend en argument `p1` et `p2` les adresses de deux entiers. La fonction échange si besoin les contenus des adresses `p1` et `p2` de façon à ce que le contenu à l'adresse `p1` soit le plus petit des deux.
2. Écrire un programme principal qui simule le lancer de trois dés à 6 faces, stocke le résultat dans les variables `d1`, `d2` et `d3` et affiche les valeurs `d1`, `d2` et `d3` (on utilisera la fonction `generer_entier` vue précédemment). Le programme utilise ensuite la fonction `ordonne` pour échanger si besoin les valeurs de façon à avoir `d1` $\leq$ `d2` $\leq$ `d3`. Pour finir, il affiche la valeur des trois dés dans l'ordre croissant et indique, avec un message clair, si on a gagné au jeu du 421 (avoir exactement un 4, un 2 et un 1).

Solution :

1. ```
   #include <stdio.h>

   void ordonne(int *p1,int *p2){
       if (*p1>*p2){
   ```
   ```

```
            int tmp = *p1;
            *p1  = *p2;
            *p2  = tmp;
        }
    }

    int main(){
        int a = 43;
        int b = 38;
        ordonne(&a,&b);
        printf("a=%d, b=%d\n",a,b);
        return 0;
    }
```

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>

    void ordonne(int *p1,int *p2){
        if (*p1>*p2){
            int tmp = *p1;
            *p1  = *p2;
            *p2  = tmp;
        }
    }

    int main(){
        srand(getpid());
        int d1 = rand() % 6 + 1;
        int d2 = rand() % 6 + 1;
        int d3 = rand() % 6 + 1;

        printf("Avant ordonnancement: %d%d%d\n",d1,d2,d3);

        ordonne(&d1,&d2);
        ordonne(&d2,&d3);
        ordonne(&d1,&d2);

        printf("Apres ordonnancement: %d%d%d\n",d1,d2,d3);

        if ( d1 == 1 && d2 == 2 && d3 == 4)
            printf("Gagné!\n");

        return 0;
    }
```

Exercice 3 :

Un tableau à deux dimensions (2D) est un tableau dont les cases sont référencées par deux indices. Ses deux dimensions sont le nombre de lignes et le nombre de colonnes du tableau. De même que pour un tableau à une dimension, en C, les cases d'un tableau 2D sont toutes du même type. Voici comment on déclare un tableau, par exemple pour un tableau d'entiers de 5 lignes et 3 colonnes :

2. `int nomTab[5][3];`

On accède à un élément à l'aide de ses indices de ligne et de colonne : `nomTab[i][j]` correspond à la case située à la ligne d'indice `i` et à la colonne d'indice `j`. Attention, la numérotation commence à 0, donc par exemple `nomTab[2][1]` désigne la case située à la 3ème ligne, 2ème colonne.

1. On considère un tableau 2D d'entiers à 10 lignes et 15 colonnes. Dans ce tableau, il n'y a que des 0 et des 1. Écrire une fonction `nb_voisins` qui prend en argument un tel tableau `tab`, un indice de ligne `lig` et un indice de colonne `col` et qui renvoie le nombre de 1 présents dans les voisins de la case d'indice `(lig,col)`. On considérera que deux cases sont voisines si elles ont un côté ou un angle en commun (donc maximum 8 voisines par case).
2. Écrire une deuxième version de cette fonction en considérant cette fois que la dernière ligne est adjacente à la première et que la dernière colonne est adjacente à la première (ainsi toutes les cases ont bien exactement 8 voisines).

Solution :

1.
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int nb_voisins(int tab[10][15],int ligne,int colonne){
    int voisins = -tab[ligne][colonne];

    int minLigne = -1;
    int maxLigne =  1;
    int minColonne = -1;
    int maxColonne =  1;

    if (ligne==0)
        minLigne = 0;
    if (ligne==10)
        maxLigne = 0;
    if (colonne==0)
        minColonne = 0;
    if (colonne==10)
        maxColonne = 0;

    for(int i=minLigne;i<=maxLigne;i++)
        for(int j=minColonne;j<=maxColonne;j++)
            voisins += tab[ligne+i][colonne+j];

    return voisins;
}

int main(){
    int monTab[10][15];

    srand(getpid());
    for (int i=0;i<10;i++)
        for (int j=0;j<15;j++)
            monTab[i][j]=rand() % 2;

    for(int i=0;i<10;i++){
        for(int j=0;j<15;j++)
            printf("%d ",monTab[i][j]);

        printf("\n");
    }

    printf("Nombre de voisins de (4,0): %d\n",nb_voisins(monTab,4,0));

    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int nb_voisins2(int tab[10][15],int ligne,int colonne){
    int voisins = -tab[ligne][colonne];

    int minLigne = -1;
    int maxLigne =  1;
    int minColonne = -1;
    int maxColonne =  1;

    for(int i=minLigne;i<=maxLigne;i++)
        for(int j=minColonne;j<=maxColonne;j++)
            voisins += tab[(ligne+i) % 10][(colonne+j) % 15];

    return voisins;
}

int main(){
    int monTab[10][15];

    srand(getpid());
    for (int i=0;i<10;i++)
        for (int j=0;j<15;j++)
            monTab[i][j]=rand() % 2;

    for(int i=0;i<10;i++){
        for(int j=0;j<15;j++)
            printf("%d ",monTab[i][j]);

        printf("\n");
    }

    printf("Nombre de voisins de (4,0): %d\n",nb_voisins2(monTab,4,0));

    return 0;
}
```

Exercice 4 :

Des chaînes de caractères... écrire les fonctions :

1. `copie_chaine` qui, étant donné deux chaînes de caractères, copie la seconde dans la première (on suppose que la première chaîne est suffisamment grande pour recevoir la copie).
2. `index` qui, étant donné une chaîne de caractères et un caractère, renvoie un pointeur vers la première occurrence du caractère dans la chaîne ou `NULL` s'il ne s'y trouve pas.
3. `index_a_rebours` qui, étant donné une chaîne de caractères et un caractère, renvoie un pointeur vers la dernière occurrence du caractère dans la chaîne ou `NULL` s'il ne s'y trouve pas. Peut-on écrire `index_a_rebours` en utilisant `index` ?
4. `duplique_chaine` qui, étant donné une chaîne de caractères, renvoie une copie de cette chaîne. A la différence de la fonction de la question 1, cette fonction devra allouer suffisamment de mémoire pour stocker la copie.

Solution :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void copie_chaine(char *a,char *b){
    int i=0;
    while (b[i]!='\0'){
        a[i]=b[i];
        i++;
    }
    a[i]='\0';
}

char* index_(char *a,char l){
    int i=0;
        while (a[i]!='\0'){
                if (a[i]==l)
                    return &a[i];
        i++;
        }
    return NULL;
}

char* index_a_rebours(char *a,char l){
    int i=0;
    while (a[i]!='\0'){
        i++;
    }
    while(i>0){
        if (a[i]==l)
            return &a[i];
        i--;
    }
    return NULL;
}

char* index_a_rebours2(char *a,char l){
    char *p=a;
    int loop_once = 0;
    while (index_(p,l)!=NULL){
        p = index_(p,l);
```

```
        p++;
        loop_once = 1;
    }
    if (loop_once>0)
        return --p;
    return NULL;
}

char* duplique_chaine(char *a){
    int taille = 0;
    while (a[taille]!='\0'){
                taille++;
        }
    char *copie = malloc(taille*sizeof(char));

    for (int i=0;i<taille;i++)
        copie[i]=a[i];

    return copie;
}

int main(){
    char a[10]="Hello!";
    char b[10]="Bye!";
    char c[10]="Hello!";

    copie_chaine(a,b);

    printf("a=\"%s\"\n",a);

    printf("Adresse=%p\n",index_(c,'l'));
    printf("Adresse a rebours=%p\n",index_a_rebours(c,'l'));
    printf("Adresse a rebours=%p\n",index_a_rebours2(c,'l'));

    printf("Chaine dupliquée=%s\n",duplique_chaine(c));
    return 0;
}
```

Exercice 5 :

1. Créer une structure `fiche_de_personnel` contenant toutes les informations critiques concernant une personne : nom, prénom, taille, age, couleur préférée, pokémon favori, ...
2. Écrire une fonction `afficher_fiche` qui affiche une fiche de personnel qui lui est passée en paramètre.
3. Écrire une fonction `lire_fiche` qui lit au clavier toutes les informations relatives à une fiche de personnel et les stocke dans une fiche passée en paramètre.
   — comment la fiche dans laquelle les données sont stockées est-elle passée à la fonction ?
   — que se passe-t-il en cas d'erreur de saisie de la part de l'utilisateur (valeurs invalides par exemple) ?
   — comment gérer correctement les cas d'erreur ?
4. Écrire une fonction qui, étant donné une fiche de personnel, fabrique une chaîne de caractères contenant toutes les informations de la fiche donnée. Attention, il faudra allouer de la mémoire pour fabriquer cette chaîne.

Solution :

```
#include <stdio.h>
#include <stdlib.h>
```

```c
struct fiche_de_personnel{
    char nom[20];
    char prenom[20];
    int age;
};

void afficher_fiche (struct fiche_de_personnel fiche) {
    printf("Nom:\t%s\nPrenom:\t%s\nAge:\t%d\n",fiche.nom,fiche.prenom,fiche.age);
};

char * construire_chaine (struct fiche_de_personnel fiche) {
    char *chaine=malloc(100*sizeof(char));
    sprintf(chaine,"Nom: %s\nPrenom: %s\nAge: %d\n",fiche.nom,fiche.prenom,fiche.age);

    return chaine;
};

void lire_fiche(struct fiche_de_personnel * fiche){
    printf("Nom?:\t\t");
    scanf("%s",fiche->nom);
    printf("\nPrenom?:\t");
    scanf("%s",fiche->prenom);
    printf("\nAge?:\t\t");
    scanf("%d",&(fiche -> age));
    // Ou sinon, pour contrôler mieux les choses
    // char age_str[3];
    // scanf("%s",age_str);
    // if (sscanf(age_str,"%d",&(fiche->age))!=1)
    //  return 1
    printf("\n");
};

int main(){

    struct fiche_de_personnel professeur = {"Couillet","Romain",39};

    afficher_fiche(professeur);

    struct fiche_de_personnel nouveau;
    lire_fiche(&nouveau);
    afficher_fiche(nouveau);

    printf("\n%s\n",construire_chaine(nouveau));
    return 0;
};
```

Exercice 6 :

Quelques fonctions bonus autour des chaînes de caractères et des structures

1. `chaine_vers_entier` qui, étant donné une chaîne de chiffres décimaux, renvoie l'entier représenté par cette chaîne. Par exemple `chaine_vers_entier("203")` renvoie 203. Si un des caractères de la chaîne n'est pas un chiffre, la fonction renvoie `-1`.
2. Utiliser la fonction `generer_entier` vue précédemment pour écrire un programme qui crée un tableau de n fiches de personnel, avec n généré aléatoirement, le remplit d'un contenu aléatoire et affiche son contenu.

Solution :

1.
```c
#include <stdio.h>

int chaine_vers_entier(char *chaine){
    int entier = 0;
    int chiffre;
    while(chaine[0]!='\0'){
        chiffre = chaine[0]-48;
        if (chiffre<0 || chiffre>9)
               return -1;
        entier = 10*entier+chiffre;
        chaine++;
    }
    return entier;
}

int main(){
    printf("Entier: %d\n",chaine_vers_entier("203"));
    return 0;
}
```

2. On reprend les codes de l'Exercice 5 et on adpate par exemple le `main` ainsi (ici les noms et prénoms sont des chiffres tirés au hasard)

```c
int main(){

        int n = generer_entier(100);
        struct fiche_de_personnel fiches[n];
        for (int i=1;i<=n;i++){
                sprintf(fiches[i].nom,"%ld",generer_entier(10000));
                sprintf(fiches[i].prenom,"%ld",generer_entier(10000));
                fiches[i].age = generer_entier(100);
        }

        for (int i=1;i<=n;i++)
                afficher_fiche(fiches[i]);

        return 0;
};
```